



# Expanding Blue Chameleon : Add-On Development Documentation



November 14, 2013

# Chapter 2

## A guide to developing your Blue Chameleon Add-On

Blue Chameleon brings company management concept to a higher plane by giving you the power to develop the add-on that suits your needs. You will be able to create your own screens, forms, data elements... and act on them as a full-fledged part of your Blue Chameleon account.

The scripting language OSL (see Chapter 1), though better suited to be handled by IT people already familiar with programming concepts, database manipulation and HTML layout, still remains simple, straightforward and quick to learn.

### 2.1 The structure of a Blue Chameleon Add-on

Inside Blue Chameleon, an add-on is made of different components :

- a **.pht library**, resulting from the compiling (2.2.1) of individual **.pht** files written in Blue Chameleon scripting language OSL ;
- files such as **html** files, Javascript **.js** files, custom style sheets **.css**, images...

The **.pht** library and the other files are to be uploaded (2.2.2) onto their dedicated directories, respectively called the **SysLibHome** and the **PublisherHome** :

```

AddOn.pht          → SysLibHome :
                   → /srv/www/osl/OSS/[YourShopName]/

Header.html        → PublisherHome :
processStuff.js    → /srv/www/htdocs/IncShop/IncModelShop/[YourShopName]/
MyOwnStyle.css     → /srv/www/htdocs/IncShop/IncModelShop/[YourShopName]/css/
Banner.jpg         → /srv/www/htdocs/IncShop/IncModelShop/[YourShopName]/images/
                   ...
```

In the above, [YourShopName] is the alphanumerical, 16-character string featured in your shop login page URL (http://www1.inc.lu/IncShop/IncModelShop/[YourShopName]/osslogin.htm).

### 2.1.1 How .phs pages are interpreted and displayed

The following address displays for instance the "Default.phs" page belonging to a compiled *MyLibrary* library :

```
http://www1.inc.lu/Scripts/sql.exe?SqlDB=[YourShopName]&Sql=MyLibrary:Default.phs
&xid=123...&Var1=&Var2=...
```

It is made of the following elements :

- the Blue Chameleon shop server, calling the `sql.exe` script (which interprets the OSL language) ;
- a string of CGI variables, amongst which three are essential :
  - `SqlDB`, which identifies your shop ;
  - `Sql`, which is the page that is displayed, always in the form `LibraryName:Page.phs` ;
  - `xid`, which is an alphanumerical string (automatically generated upon login) identifying your session.

## 2.2 Basics of add-on developing : how it is integrated

Generally speaking, as summed up on Fig.2.1, the process of developing your own add-on will consist in :

- write the .phs files that constitute it (mandatorily including a `ossModuleRegister.phs` one, see 2.3.1.1) ;
- compile them into a compiled library (`AddOn.phs`) ;
- upload this library to your shop's *SysLibHome* ;
- register your library and create related menu element (this has only to be done the first time).



*For add-on to be registered, it must contain a `ossModuleRegister.phs` script (2.3.1.1).*

The add-on has to be registered first and inserted as a menu, before being accessible. After, developing will consist only in generating the library and uploading it.



*It is to note that file names (including the .phs extension) should not be greater than 23 characters.*

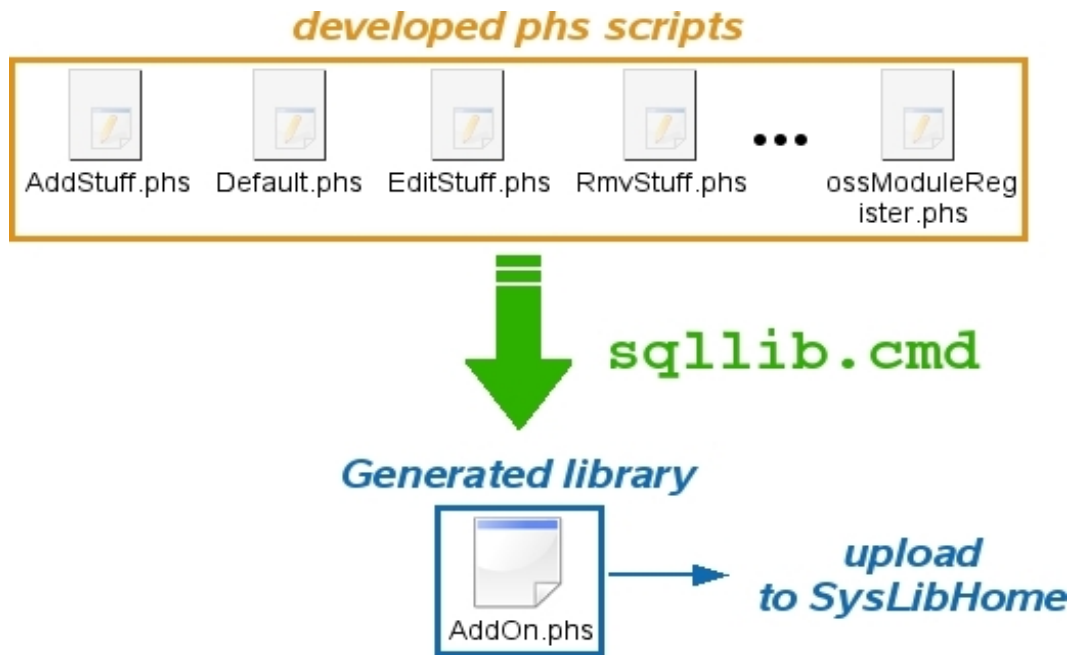


Figure 2.1: A single compiled library `AddOn.pht` is generated and is to be uploaded unto the *SysLibHome*.

## 2.2.1 Compiling your .pht files

All of these files need to be placed in the same directory. Then, the `sqllib.cmd` (or `sqllib.sh`) executable will generate the compiled library `AddOn.pht` (final name can be set by modifying executable).

## 2.2.2 Uploading your library and files

Uploading of compiled library and, if applicable, of other files, is done respectively in the *SysLibHome* and *PublisherHome* directories as cleared up in 2.1.

## 2.2.3 Registering your add-on

**Registering your add-on inside your shop - as well as menu-related concerns - has only to be done once.**

Inside the Blue Chameleon environment, the add-on you develop is called a 'custom library', which has to be registered as such. The process of doing so is described at Fig.2.2.

It has to be emphasized that the name as entered in the 'Library:' field must correspond exactly to the uploaded library's name, minus the `.pht` extension.

The 'Label:' can be entered as a multilingual string (2.3.3).

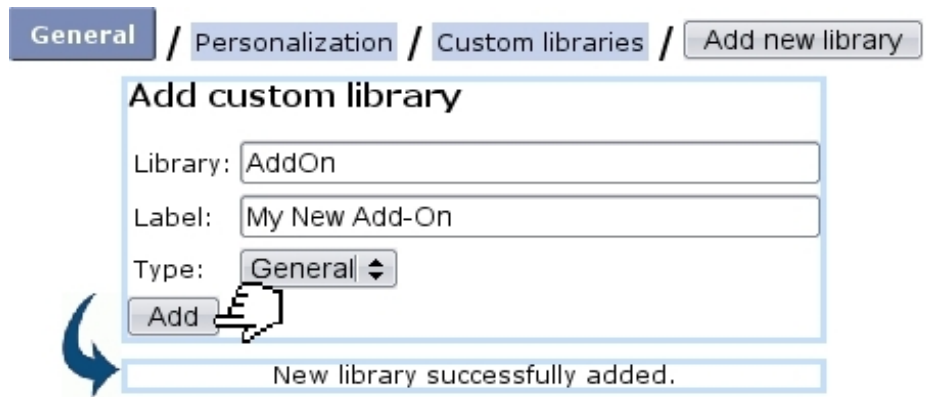



Figure 2.2: The compiled library AddOn.phs is added.

### 2.2.3.1 Setting a new menu script and a menu element for the add-on



*User rights for menu script management can be set up either at the Modify User Page or the User Right Page under the element*

**Script management**

Ideally, your add-on would be accessed through the menu. For this to be possible, a 'menu script' has to be created. It will link to the *entry point*, which is the page of your add-on you want to be displayed when accessing it, for instance `Default.phs`. Fig.2.3 features how a new menu script is added, provided that the custom library has been well registered as described above.

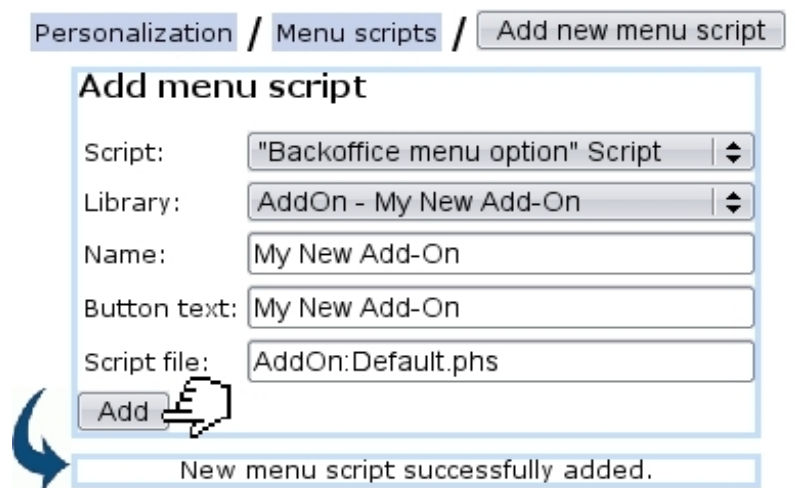


Figure 2.3: This enables to make the new add-on accessible (more precisely, its `Default.phs` script) from menu configuration.

It is to note that the "Script file:" as featured there must always be entered this form : add-on name, colon and script file. The 'Button text:' can be entered as a multilingual

string (2.3.3).

Once created, as shown in Fig.2.4, the add-on name (as typed in the 'Button name:' field) now appears in the list of available procedures while at the Menu Configuration context.

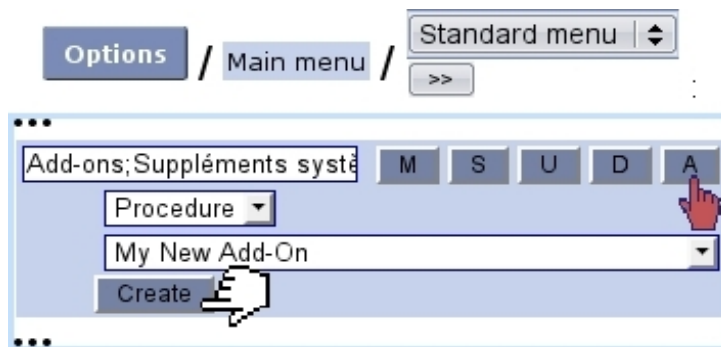


Figure 2.4: The add-on is indeed now available to be added as a menu element.

Nonetheless, for the new add-on to appear at the menu (here through **Add-ons**), the menu script user rights to the add-on must be set.

### 2.2.3.2 Setting the add-on's access rights

As shown on Fig.2.5, the add-on's access rights are now featured on the *Modify User Page* under [*Menu script user rights* :]. When they are set sufficiently, the new add-on can be accessed.

## 2.3 Basics of add-on developing : how to organize your .pht scripts

Fig.2.6 illustrates in a general way how an add-on could be ideally structured. An add-on is constituted of a "main" .pht file, most conveniently named `Default.pht` and which is the entry point when registering the library (2.2.3.1).

As the add-on is developed through time, other script files are created. They might be called by `Default.pht` or between each other, by the way of `INCLUDEs` (1.4.1) and forms (2.4.3.2).

Also, it is to note that "outer" scripts, especially belonging to the `ossbo` library, may have to be called, for inserting header/footer (2.3.2), getting user rights (2.5.1)... `ossbo` is Blue Chameleon's main library.

As for a .pht script itself, it may be composed most conveniently along general guidelines, as shown in Fig.2.7.

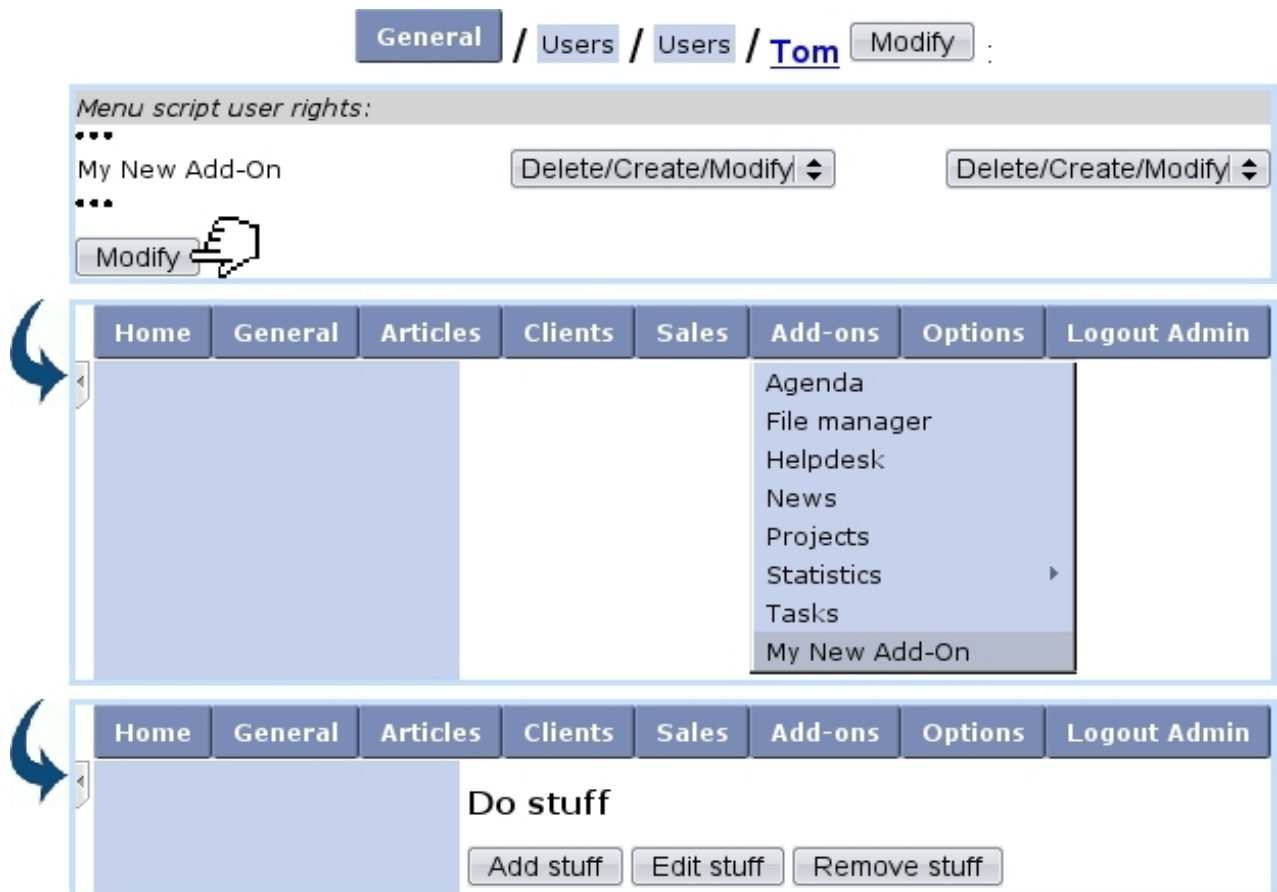



Figure 2.5: Access user rights for the new add-on, initially set by default to 'None', are raised up for user Tom and add-on thus becomes accessible.

### 2.3.1 "oss-" scripts : the basics

These are scripts that will be recognized and used by Blue Chameleon to provide various things such as the registering of your library (`ossModuleRegister.php`, mandatory), dedicated user rights (`ossUserConfig.php`, `ossUserModify.php`)...

The oss- scripts that your add-on (or any library, for that matter) has been equipped with can simply be seen checked as seen in Fig.2.8.



*For a new oss- script to be registered, it is necessary to 'recheck custom libraries' via*

**General** / Personalization / Check custom libraries .

More about oss- scripts can be found at 2.10.

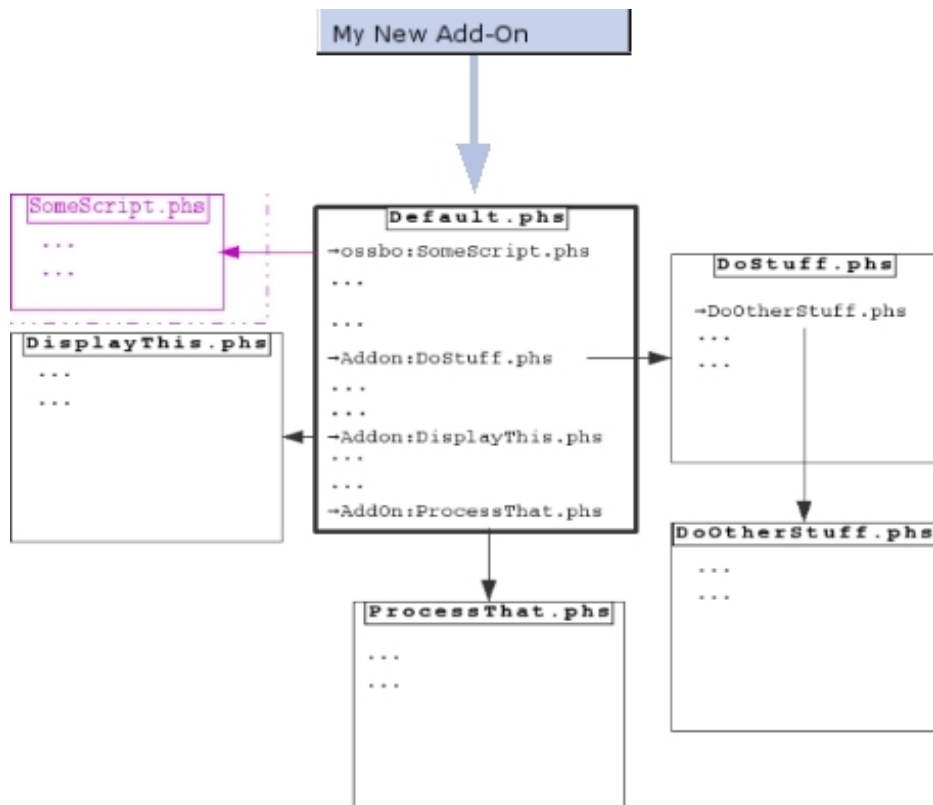


Figure 2.6: An example of the branching between .php files in an add-on.

### 2.3.1.1 ossModuleRegister.php

This script, which has to be included amongst your set of .php files, simply consists in one line :

```
«RETURN 1003»
```

"1003" there is an example, any integer over 1000 can be taken. This number is called the *signature* of the *library* and it is used to identify it (see Fig.2.8).

### 2.3.1.2 ossUserConfig.php, ossUserModify.php

How to compose these library-right-generating scripts is explained in detail at *Making custom library user rights for your add-on* (2.5.3).

## 2.3.2 Headers and footers

In order to have your add-on displayed in the same style than the rest of Blue Chameleon, it is advised to include, around the beginning, the "Header.php" script (belonging to library *ossbo*), in the following way :

```
«VAR NEW _Title="My page title;Mon titre de page;Mein Seitentitel"»
«INCLUDE ossbo:Header.php;_Title=_Title;_Style="print";»
```



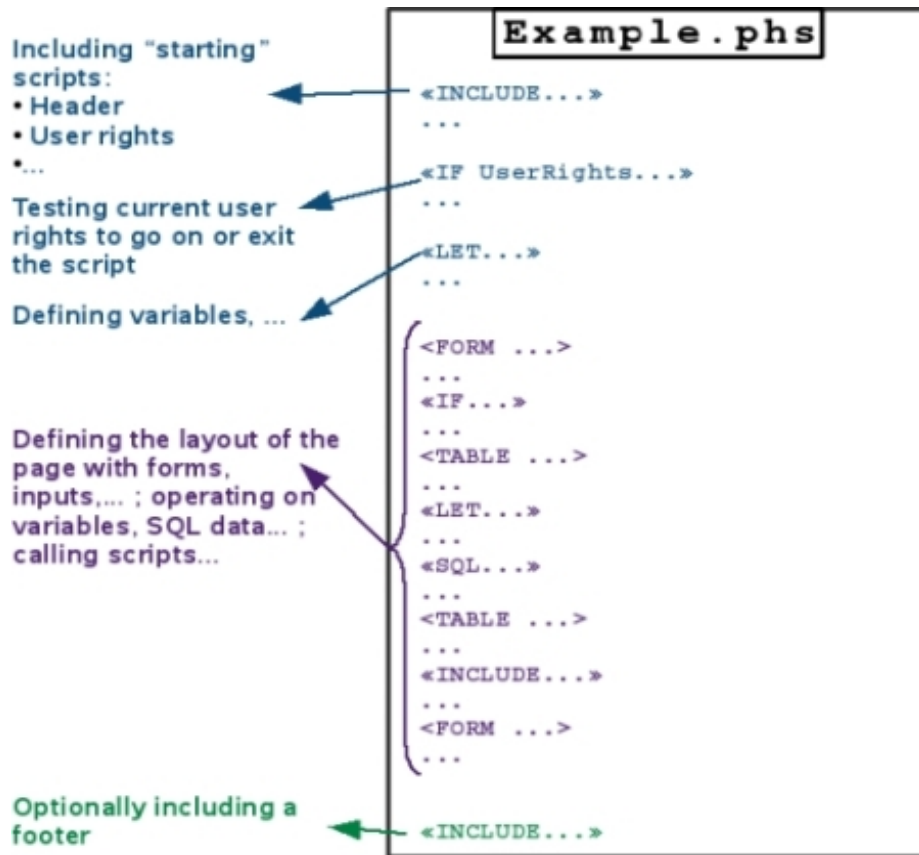


Figure 2.7: A .php file may be composed in this rather practical way.

General / Personalization / Custom libraries / **AddOn - AddOn** :

**View custom library**

Library: AddOn  
 Signature: 1003  
 Label: AddOn  
 Type: General

**Available scripts:**

•••  
 ossUserConfig OK  
 •••  
 ossUserModify OK

Figure 2.8: oss- scripts that the add-on contains. The presence of `ossModuleRegister.php` is confirmed by the existence of a 'Signature:'.

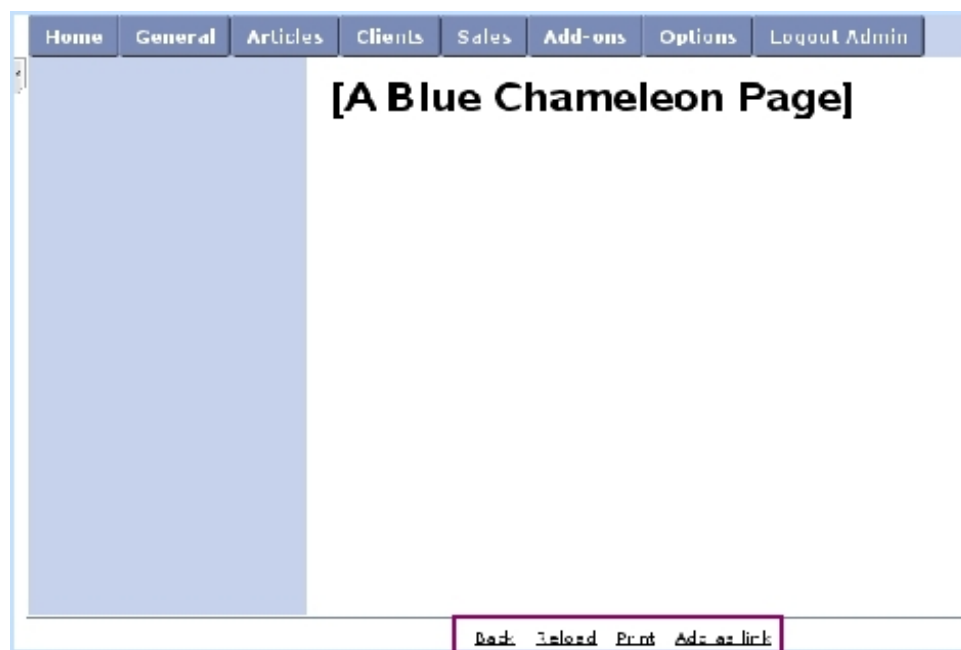
with :

- `_Title` being the title you want to give to the page generated by the script, as displayed in the top of the browser's window. It can be entered as a multilingual string (2.3.3) ;

- `_Title="print"` calling for Blue Chameleon's usual `css` style sheet, but it is possible to use your own, for instance `_Style="MyOwnStyle"` (provided that a `MyOwnStyle.css` has been uploaded to *PublisherHome/css*).

As for the footer, as included via `«INCLUDE ossbo:Footer.phs»`, it allows to display the links as featured in Fig.2.9, enabling, for this page :

- to go back to the previous one ;
- to reload it ;
- to print it ;
- to add it as a User Menu link (see Blue Chameleon Documentation, **General Interface Handling**, *Link management*).



**Back - Reload - Print - Add as link**

Figure 2.9: These four links are enabled by script `ossbo:Footer.phs`.

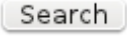


These header and footer are to be included in the script only if it displays a new page ; it is unnecessary to feature them in scripts that do not display anything on screen, nor in `INCLUDE`'d scripts.

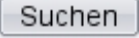
### 2.3.3 Languages

As Blue Chameleon can handle different languages that you enable while at the *Merchant Configuration Page*, you might want to make sure that - especially if users have selected different preferred languages -, any displayed text element of your add-on is provided through a *multilingual string*, i.e. as an example `My Text;Mon Texte;Mein Text;...`

This multilingual string, when it has to be displayed, is then interpreted with the `WRITE LANGUAGE` command.

For instance, while the following code would display a  button only in English :

```
<input type="submit" name="Search" value="Search">
```

...this code below, on the other hand, would display  if user's preferred language were German :

```
<input type="submit" name="Search"
value="«WRITE LANGUAGE Search;Chercher;Suchen»">
```

Therefore, if you wish to allow any piece of text to be displayed according to user's preferred language, you should replace it by command `«WRITE LANGUAGE My Text;Mon Texte;Mein Text;...»` for as many languages that you have enabled at the *Merchant Configuration Page*. The order of languages is as follows :

Language	Value
English	1
French	2
German	3
Dutch	4
Portuguese	5
Luxembourgish	6
Spanish	7
Italian	8
Swedish	9
Polish	10
Hungarian	11
Russian	12

Function `@languageitem` also performs the same duty :

```
«=@languageitem("Search;Chercher;Suchen")»
```

If multilingual string is to be used in an Ajax environment, display issues might arise if special characters or accented letters are involved : it is then advised to use function `@atohtml2` (1.10.6.2), or more directly the following syntax :

```
«WRITE LANGUAGE %html;Next;Précédent;Nächst»
```

or also

```
«%html;=@languageitem("Next;Précédent;Nächst")»
```

This %html; right a string directly performs @atohtml2's action.

When Javascript dialog boxes are involved, the @atojavascript (1.10.6.3) should be used in order to display multilingual strings with non-ASCII characters correctly. It is also available in a short form :

```
<script language="JavaScript">
alert("«WRITE LANGUAGE %js;Item is not available;Objet non disponible;Produkt
      ist nicht verfügbar»")
...

```

## 2.4 Variables, forms, scripts

### 2.4.1 CGI variables

As previously explained (*How .phs pages are interpreted and displayed*, 2.1.1), 3 CGI variables (SqlDB, Sql and xid) are *a minima* necessary to display a Shop page. They are completed by two others :

- `_MerchantId`, which is the identifier for your Merchant ;
- `_CustomScriptId`, which identifies the add-on, script-wise.

The value of these need not to be known explicitly nor modified, except for `Sql`, which rules what `.phs` script is displayed, and thus subject to be changed.

As scripts are called (2.4.3.2), **other CGI variables can be created and used.**

### 2.4.2 Accessing names and values of CGI variables

For debugging purposes, the names and values of current CGI variables are accessed via `CGINAME` and `CGIVALUE`.

In order to go through all of them, these are used within a «LIST»...«/LISTCGI» loop environment, in which instructions are executed as many times as there are CGI variables.

For instance, the following codes can be used to list all CGI variables (and their values) as used in the current script :

```
<TABLE>
  «LIST»
  <TR><TD>«CGINAME»:</TD><TD>«CGIVALUE»</TD></TR>
  «/LIST CGI»
</TABLE>
```

### 2.4.3 CGI variables and HTML forms

Whenever a HTML form is defined in a .php script, CGI variables need to be recalled as INPUTs, as written in Code 1.

---

**Code 1** Basic code to create HTML forms.

---

```
<FORM METHOD="post" ACTION="/Scripts/sql.exe">
  <INPUT TYPE="hidden" NAME="SqlDB" VALUE="«SqlDB»">
  <INPUT TYPE="hidden" NAME="Sql" VALUE="«Sql»">
  <INPUT TYPE="hidden" NAME="xid" VALUE="«xid»">
  <INPUT TYPE="hidden" NAME="_CustomScriptId" VALUE="«_CustomScriptId»">
  <INPUT TYPE="hidden" NAME="_MerchantId" VALUE="«_MerchantId»">
  [Form contents]
</FORM>
```

---

As it can be seen, the value of these is accessed like any other OSL variable, i.e. by putting them between guillemots.

Recalling the five basic CGI variables, this is the most basic form that can be created : but other CGI variables can be created within a form, as shown below.

#### 2.4.3.1 Creating CGI variables in a form

Simply enough, a new CGI variable as used in a form can be created as an INPUT :

```
<INPUT TYPE="hidden" NAME="MyCGIVar" VALUE="«MyCGIVarValue»">
```

This of course requires that, somewhere inside the form, variable MyCGIVar is actually assessed a value.

Various types of input (hidden, text, checkbox, radiobutton) can be used.

#### 2.4.3.2 Form calling another script

In Code 1, line

```
<INPUT TYPE="hidden" NAME="Sql" VALUE="«Sql»">
```

means that, upon generating of this form, the current script (which name always is accessible via «Sql») was called : thus, upon submit, system remained on the current script.

On the other hand, if we wanted the form to call another script, named for instance AddStuff.php script, this line would have been replaced by :

```
<INPUT TYPE="hidden" NAME="Sql" VALUE="AddOn:AddStuff.php">
```

It has to be noted that a script is always referred to as this way, *LibraryName:ScriptName.phs*.

Code 2 then shows what could be done to make appear an  button that, upon click, calls the `AddStuff.phs` script.

---

**Code 2** A HTML form that calls another script ; other lines are similar to Code 1.

---

```
<FORM METHOD="post" ACTION="/Scripts/sql.exe">
...
<INPUT TYPE="hidden" NAME="Sql" VALUE="AddOn:AddStuff.phs">
...
<INPUT TYPE="submit" NAME="Add" VALUE="Add stuff">
...
</FORM>
```

---

Inside the called script, any CGI variable as defined within the previous form then exists, and its value is accessible with the usual `«...»`.

#### 2.4.4 Conserving variables from one script to another

There are different ways to branch `.phs` scripts :

- by means of form submission, which conserves only CGI variables defined therein ;
- by means of an `«INCLUDE... »`, thanks to which any variable as used before exists in the included script (this is used for instance to initialize new variables before an included script that will assess values to them, see for instance *Information on users* 2.6.1.2) ;
- by means of a redirection.

In the latter case, a string containing all basic 5 CGI variables, as well as other variables you want to pass, must be built in the following fashion :

```
...
...
...
<META HTTP-EQUIV="refresh" CONTENT="0;
URL=/Scripts/sql.exe?SqlDB=«SqlDB»&Sql=AddOn:RedirScript.phs&xid=«xid»
&_MerchantId=«_MerchantId»&_CustomScriptId=«_CustomScriptId»
&_Var1=«_Var1»&_Var2=«_Var2»"&...>
```

## 2.4.5 CGI variables and javascript functions

It can happen that the *same* form provides different possible actions, for instance several buttons that would call each for a different script. In order to implement this, Code 3 provides a solution using simple javascript functions ; inside this example form :

- CGI input for Sql does not call for any script ;
- two  and  buttons (for instance) call for two javascript functions `onModify()` and `onDelete()` as shown in Code 4.

---

**Code 3** This form contains buttons that, on click, call for javascript functions outside the form.

---

```
<FORM METHOD="post" ACTION="/Scripts/sql.exe">
...
<INPUT TYPE="hidden" NAME="Sql" VALUE="">
...
<INPUT TYPE="button" VALUE="Modify" onClick="onModify()">
...
<INPUT TYPE="button" VALUE="Delete" onClick="onDelete()">
...
</FORM>
```

---

The `onModify()` and `onDelete()` functions then simply consist in assigning the right script name (`AddOn:Modify.phs` or `AddOn>Delete.phs`) to CGI variable `Sql`.

## 2.4.6 Incorporating other files

In order to make the layout of a `.phs` script less cluttered, the *PublisherHome* (2.1) can be used to store various files such as javascript files, custom style sheets... This is done as shown in Code 5 by using the predefined variable `«#SQLWWWHOME#»` which stores the *PublisherHome* root.

## 2.5 Setting aimed user rights for the add-on



*For general information about user rights, please check User right basics in the Annex of Blue Chameleon's Extended or Full Documentation.*

User rights' purpose is that to restrict certain actions that, when done carelessly, could harm the data the add-on deals with.

---

**Code 4** How javascript functions for calling a .php script are made .

---

```
<script language="JavaScript">

    function onModify(){
        document.edit.Sql.value="AddOn:Modify.php";
        document.edit.submit();
    }

    function onDelete(){
        Cfrm_Box=confirm("<WRITE LANGUAGE Are you sure;Etes-vous sûr;Sind Sie
sicher>?");
        if (Cfrm_Box==true){
            document.edit.Sql.value="AddOn:Delete.php";
            document.edit.submit();
        }
    }
}

</script>
```

---

---

**Code 5** Including files "processStuff.js" and "MyOwnStyle.css" from their respective locations js/ and css/ on *PublisherHome* root.

---

```
<script type="text/javascript" src="<#SQLWWWHOME#>/js/processStuff.js">
</script>

<link rel="stylesheet" href="<#SQLWWWHOME#>/css/MyOwnStyle.css"
type="text/css" media="screen"/>
```

---



## 2.5.1 Getting the current (Menu Script) user rights inside a script

During the insertion of the add-on, dedicated user rights for it ("Menu script user rights") are indeed created (2.2.3.2) : they may rule in a simple what can or cannot be done inside the add-on.

For a .phs script to know what these current rights for your add-on are, it must INCLUDE the `ossbo:OSSCustomUserRights.phs` script as shown in Code 6. Indeed, it uses as a parameter the CGI variable (2.4.1) `_CustomScriptId` that identifies your add-on menu script-wise and stores the right values (strings) into two previously initialized variables, one for the owner's user rights and the other for "All".

---

**Code 6** This code stores in `_OwnerUserRights` and `_UserRights` the values of current **Menu Script user rights** for the add-on.

---

```
«VAR NEW _OwnerUserRights=""»
«VAR NEW _UserRights=""»

«INCLUDE ossbo:OSSCustomUserRights.phs;_MerchantId=«_MerchantId»;
_CustomScriptId=«_CustomScriptId»»
```

---

This code must be placed at the top of *any* script in which the values of menu script user rights are going to be needed ; this is not necessary if `_OwnerUserRights` and `_UserRights` are carried around from script to script as CGI variables.

### 2.5.1.1 User rights values

With Code 6 included at the beginning of a script, add-on user rights for "All" have been recuperated and can be accessed via `_UserRights[0]`, which stores a value that increases as user's rights become more extended, according to the following :

Value of <code>_UserRights[0]</code>	Corresponding user right
48	None
49	View
50	Modify
51	Create/Modify
52	Delete/Create/Modify
88	Unrestricted

(These values correspond in fact to the ASCII value of characters 0, 1, 2, 3, 4 and X.)

For instance, for a user that has been set with Create/Modify rights for the add-on (at her/his *Modify User Page*) estimating `_UserRights[0]` will return 51.

This now can be used in practice to restrict the access to the add-on's features, as explained below.

## 2.5.2 Fine-tuning the access to add-on's features

Inside your add-on, data might have to be modified, other data to be created while deleting some might also come around ; some of these actions will never be performed by certain users.

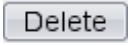
### 2.5.2.1 Access to elements of a script

Whenever an element of an add-on's script shall not appear unless user rights are sufficient enough to allow it, it can be put between `«IF»...«ENDIF»` tags, with a test done on `_UserRights[0]` : for instance, the following code

```
«IF _UserRights[0]>51»
  [Instructions]
«ENDIF»
```

will process `[Instructions]` only if user has at least Delete/Create/Modify menu script user rights. It can therefore be used for instance (taking the example as used in 2.4.5 for javascript functions) to hide data-deleting features as shown on Code 7.

Thanks to this way of testing for user rights, the following is achieved :

- the  button only appears for users who have at least Delete/Create/Modify rights. Therefore, this call to the `onDelete()` function is *de facto* disabled for users who have insufficient rights ;
- the fact that the `onDelete()` function is also *itself* protected in a similar way is a further protection ensuring that, if ever "unprotected" calls to it are done, they will abort.

### 2.5.2.2 Access to a whole script

As scripts multiply, it becomes sometimes uneasy to tell if a particular script, access to which was supposed to be protected, is indeed accessed under sufficient user rights.

To ensure that no "trespass" will happen, as well as to simply render a script restricted under other circumstances, the command `EXIT STOP` can be used to stop the execution of the script. For instance, the Code 8, placed at the beginning of the script (after user rights have been recuperated), tests if user has at least Modify rights and aborts the script if not, displaying the "You don't have the necessary rights to access this function." message.

---

**Code 7** These elements are accessed only if `_UserRights[0]` corresponds to at least Delete/Create/Modify.

---

```
<FORM METHOD="post" ACTION="/Scripts/sql.exe">
```

```
...
```

```
«IF _UserRights[0]>51»  
  <INPUT TYPE="button" VALUE="Delete" onClick="onDelete()">  
«ENDIF»
```

```
...
```

```
</FORM>
```

```
...
```

```
<script language="JavaScript">
```

```
...
```

```
«IF _UserRights[0]>51»  
  function onDelete(){  
    ...  
  }  
«ENDIF»
```

```
...
```

```
</script>
```

---

---

**Code 8** This restricts the access to the rest of the script to any user who has insufficient rights.

---

[Code 6]

```
«*Is user allowed to go further ?*»  
«IF _UserRights[0]<50»  
  «INCLUDE ossbo:MessageAccessDenied.phs#Message»  
  «EXIT STOP»  
«ENDIF»  
...
```

---

## 2.5.3 Making custom library user rights for your add-on

In the above, the user rights that were recuperated as described in 2.5.1 were the "Menu Script User Rights" : a single value for each user was obtained. As an add-on expands, it may eventually nonetheless require more than one value.

Blue Chameleon provides the possibility to create new - library - user rights for your add-on. In order to do this, follow this guideline :

1. open two files called `ossUserConfig.phs` and `ossUserModify.phs` ;
2. paste the Codes 9 and 11 ;
3. fill below in your custom user rights as shown as an example in Code 10 ;
4. compile the library.

### 2.5.3.1 Building your own `ossUserConfig.phs` script

This script's aim is to make appear, on the *Modify User Page*, your own custom user rights.

---

**Code 9** The preamble to put in `ossUserConfig.phs`.

---

```
«IF _UserId == -1»
  «VAR NEW azOwnerUserRights = "00000000"»
  «VAR NEW azUserRights = "00000000"»
«ELSE»
  «SQLEXEC STRING azUserRights = select azUserRights from
OSSCUSTOMLIBRARYUSERRIGHTS where idUser = «_UserId» AND idLibrary =
«_LibraryId»»
  «SQLEXEC STRING azOwnerUserRights = select azOwnerUserRights from
OSSCUSTOMLIBRARYUSERRIGHTS where idUser = «_UserId» AND idLibrary =
«_LibraryId»»
«ENDIF»
```

---

In Code 9, two strings of rights (for "All" and "Owner") are selected for the current user and the current library (your add-on) : they correspond to the custom rights that are going to be defined displayed further down, in Code 10.

There :

- a general title is given ("*My AddOn's Custom User Rights*") ;
- a first custom right is defined (here, "Gizmo Management"), under which are defined two list-boxes :

- one giving the usual values of user rights (None, View,... **but you can even define what you wish**), and the choice is stored in `_OwnerUserRight«_LibraryId»_0` : this will be the user right value assigned to right "Gizmo Management" concerning Owner ;
- a similar one, this time defining in `_UserRight«_LibraryId»_0` the user right value assigned to right "Gizmo Management" concerning All.
- a second custom right ("Widget Handling") is defined, similarly as for "Gizmo Management", this time with user values stored in `_OwnerUserRight«_LibraryId»_1` and `_UserRight«_LibraryId»_1` ;
- and so on, incrementing the integer as put after `_[Owner]UserRight«_LibraryId»...`

### 2.5.3.2 Building the corresponding `ossUserModify.phs` script

While the previous script made the various new right elements appear on the *Modify User Page*, a script whose code is detailed at 11 is still necessary in order to validate changes.

There :

- a first SQL command tests if the custom rights pertaining to your library have been inserted into Blue Chameleon's `OSSCUSTOMLIBRARYUSERRIGHTS` ;
- a SQL command places into this table the values of the Owner user rights for each group of rights : `«_OwnerUserRight«_LibraryId»_0»«_OwnerUserRight«_LibraryId»_1»`, that correspond respectively to "Gizmo Management" and "Widget Handling". This makes two values, and the string is thus completed by six zeros, as, in the preamble of `ossUserConfig.phs`, the variable `azOwnerUserRights` was defined with eight zeros. A similar thing is done for "All" user rights with `«_UserRight«_LibraryId»...` ;
- a last SQL command sets the rights themselves in the same way.

### 2.5.3.3 These two scripts in force

Once the `ossUserConfig.phs` and `ossUserModify.phs` files have been integrated and compiled with the library, these new user rights finally appear at the bottom of the *Modify User Page*, as featured in Fig.2.10 : they can now be set and used in your scripts, as cleared up below.

### 2.5.3.4 Incorporating your add-on custom user rights in scripts

As shown in Code 12 below, calling script `ossbo:OSSLibraryUserRight.phs` stores into strings `_OwnerUserRights` and `_GroupUserRights` (which have to be initialized before) the right vectors. The input parameters for the script, apart from the usual `_MerchantId`, are several identifiers :

---

**Code 10** An example of a `ossUserConfig.phs` file : defining various custom rights for AddOn, building for each list-boxes with multiple user right choices, for "Owner" and for "All".

---

[Code 9]

```
<tr>
  <td colspan=3><i>My AddOn's Custom User Rights</i></td>
</tr>

<tr>
  <td>Gizmo Management</td>
  <td>
    <select name="_OwnerUserRight«_LibraryId»_0">
      <option value="0" <IF azOwnerUserRights[0]==48>selected<ENDIF>>None</option>
      <option value="1" <IF azOwnerUserRights[0]==49>selected<ENDIF>>View</option>
      <option value="2" <IF azOwnerUserRights[0]==50>selected<ENDIF>>Modify</option>
      <option value="3" <IF azOwnerUserRights[0]==51>selected<ENDIF>>Create/Modify</option>
      <option value="4" <IF azOwnerUserRights[0]==52>selected<ENDIF>>Delete/Create/Modify</option>
      <option value="X" <IF azOwnerUserRights[0]==88>selected<ENDIF>>Unrestricted</option>
    </select>
  </td>
  <td>
    <select name="_UserRight«_LibraryId»_0">
      <option value="0" <IF azUserRights[0]==48>selected<ENDIF>>None</option>
      [...]
      <option value="X" <IF azOwnerUserRights[0]==88>selected<ENDIF>>Unrestricted</option>
    </select>
  </td>
</tr>

<tr>
  <td>Widget Handling</td>
  <td>
    <select name="_OwnerUserRight«_LibraryId»_1">
      [...]
    </select>
  </td>
  <td>
    <select name="_UserRight«_LibraryId»_1">
      [...]
    </select>
  </td>
</tr>
```

---

---

**Code 11** The `ossUserModify.phs` file corresponding to the `ossUserConfig.phs` as previously defined.

---

```
«SQLEXEC INT existSqlLine = select count(*) from
OSSCUSTOMLIBRARYUSERRIGHTS where idMerchant=«_MerchantId» AND
idLibrary=«_LibraryId» AND idUser=«_UserId»

«IF existSqlLine==0»
  «SQL INSERT INTO OSSCUSTOMLIBRARYUSERRIGHTS
(idMerchant,idLibrary,idUser,azOwnerUserRights,azUserRights)
VALUES(«_MerchantId»,«_LibraryId»,«_UserId»,'_«_OwnerUserRight«_LibraryId»_0»
«_OwnerUserRight«_LibraryId»_1»000000','_«_UserRight«_LibraryId»_0»
«_UserRight«_LibraryId»_1»000000')»
«ENDIF»

«SQL update OSSCUSTOMLIBRARYUSERRIGHTS set
azOwnerUserRights='«_OwnerUserRight«_LibraryId»_0»«_OwnerUserRight«_LibraryId»_1»
000000',azUserRights='«_UserRight«_LibraryId»_0»«_UserRight«_LibraryId»_1»000000'
where idMerchant=«_MerchantId» AND idLibrary=«_LibraryId» AND
idUser=«_UserId»
```

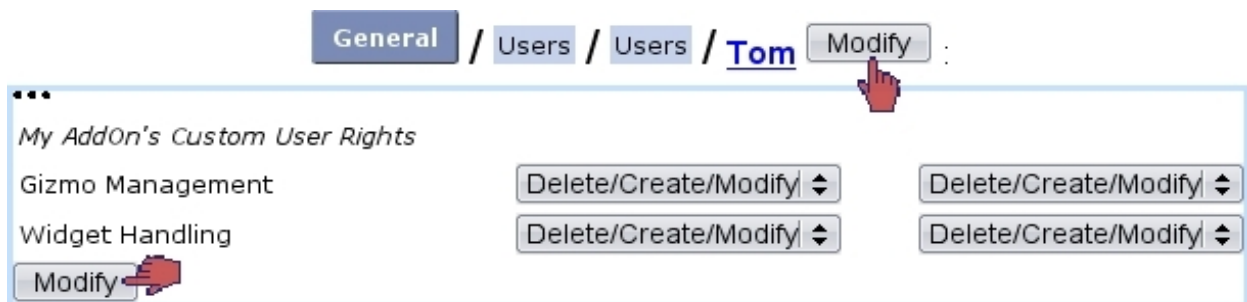


Figure 2.10: The custom rights now appear on the Modify User Page.

- `_LibraryId`, which value `«_CustomLibraryId»` corresponds to the signature (2.3.1.1) of your add-on (in the current example,1003) ;
- `_UserId`, which is set to the current user (`#SQLUSERID#`) ;
- user's `_UserGroupId`, as obtained via script `ossbo:OSSUserInfo.phs` (2.6.1.2).

Once initialized, in this current example, the `_OwnerUserRights` string will be `XX000000` for a user that has unrestricted owner rights for "Gizmo Management" and "Widget Handling". The former is then accessed via `_OwnerUserRights[0]` and the latter via `_OwnerUserRights[1]`.

The following code shows for instance how a `Gizmo Management` button can be hidden from users who do not have at least View rights for it :

---

**Code 12** Getting the custom user rights of an add-on.

---

```
«VAR NEW _UserGroupId=""»
«INCLUDE
ossbo:OSSUserInfo.phs;_MerchantId=«_MerchantId»;_UserId=«#SQLUSERID#»»

«VAR NEW _OwnerUserRights=""»
«VAR NEW _GroupUserRights=""»
«INCLUDE ossbo:OSSLibraryUserRight.phs;_MerchantId=«_MerchantId»;
_LibraryId=1003;_UserId=«#SQLUSERID#»;_UserGroupId=«_UserGroupId»»

«IF _OwnerUserRights[0]>=49 && _GroupUserRights[0]>=49»
  <INPUT TYPE="button" VALUE="Gizmo Management"
onClick="onManageGizmo()">
«ENDIF»
```



*In order to call ossbo:OSSLibraryUserRight.phs only once (at the entry script), \_OwnerUserRights and \_GroupUserRights can be conveniently transmitted from script to script as CGI variables :*

```
...&_OwnerUserRights=«_OwnerUserRights»&_GroupUserRights=«_GroupUserRights»...
```

## 2.6 Importing information on Shop data

In order to get information related to your Shop data such as users, clients, orders, invoices... Blue Chameleon's main library ossbo.phs has been equipped with a wide range of scripts aimed at fetching any kind of information as entered in other places.

Whatever you want to get, those information scripts are used in the same way :

- first, initialize output variables (only those you want to use) ;
- then, call script, using if applicable input parameters.

According to the type of the output variable to fetch, numerical value or string, it is either initialized to 0 (e.g. «VAR NEW \_BillTotalPrice=0») or to the null string (e.g. «VAR NEW \_ClientFullName="" »).

### 2.6.1 Importing user and user group information

Through the course of an add-on's development, it might come in necessary to implement functionalities that are user- or user group-dependent.

The following guideline then shows how to bring in those data.



### 2.6.1.1 List of users

The `ossbo:OSSUserList.phs` scripts can be used to return a formatted list of users.

ossbo:OSSUserList.phs		
Input Variable	Type	Corresponding data
<code>_MerchantId</code>	Integer	Merchant identifier
<code>_UserCustomScriptId</code>	Integer	Custom script identifier to test access rights [Optional]
<code>_UserMinUserRights</code>	Integer	User rights required at least [Optional]
<code>_UserGroupId</code>	Integer	User group identifier (not existing=all users, -1=users in a user group)

Return value : string `<username>=<id>,<username>=<id>,...`

#### 2.6.1.1.1 Examples of use

The following returns a list of all users :

```
«VAR NEW _UserList=""»
«INCLUDE _UserList=ossbo:OSSUserList.phs;_MerchantId=1 »
```

Returned variable `_UserList` is then formatted as "`<username>=<id>,<username>=<id>,...`" (for instance "admin=1,Dick=4,Harry=7,James=6,...").

This other example features all possible optional input parameters that this script can be called with :

```
«VAR NEW _UserList=""»
«INCLUDE _UserList=ossbo:OSSUserList.phs;_MerchantId=«_MerchantId»;
_UserCustomScriptId=«_CustomScriptId»;
_UserMinUserRights= 49; _UserGroupId=5»
```

There :

- the identifier of the custom script is passed on ;
- the 'View' user right (49) is passed on to select users that have at least this menu script user right on the add-on ;
- the identifier for the user group "Accounting" (5) is passed to select users belonging to this group.

Consequently, the `_UserList` as obtained by this call will contains only the members of the "Accounting" user group that have at least view rights on the add-on.

### 2.6.1.2 Information on users

A call to script `ossbo:OSSUserInfo.phs`, with only parameters `_MerchantId` and `_UserId`, allows to access a multitude of user-related data as filled on the *Create/Modify User Page*.

ossbo:OSSUserInfo.phs		
Input Variable	Type	Corresponding data
<code>_MerchantId</code>	Integer	Merchant identifier
<code>_UserId</code>	Integer	User identifier
Output Variable	Type	Corresponding data
<code>_UserName</code>	String	User name
<code>_UserGroupId</code>	Integer	Default user group identifier
<code>_UserGroup</code>	String	Default user group label
<code>_UserGroupList</code>	String	User group list <code>id:label;id:label...</code>
<code>_UserGroupIdList</code>	String	User group identifier list <code>id;id...</code>
<code>_UserRealname</code>	String	User name
<code>_UserFirstName</code>	String	User first name
<code>_UserTel</code>	String	User phone number
<code>_UserMobile</code>	String	User mobile number
<code>_UserEMail</code>	String	User email address
<code>_UserNotification</code>	String	User notification
Return value : <code>_Status</code> (1 if user was found, 0 if not)		

#### 2.6.1.2.1 Example of use

The following bit of script stores the current user's first name and last name as string `_CurrFullName` and displays it :

```

«VAR NEW _UserRealName=""»
«VAR NEW _UserFirstName=""»
«INCLUDE ossbo:OSSUserInfo.phs;_MerchantId=«_MerchantId»;_UserId=«#SQLUSERID#»»
«VAR NEW _CurrFullName=_UserFirstName+" "+_UserRealName»
«WRITE LANGUAGE Your are;Vous êtes;Sie sind» «_CurrFullName»

```

### 2.6.1.3 List of user groups

The `ossbo:OSSUserGroupList.phs` script returns a formatted list of user groups.

ossbo:OSSUserGroupList.phs		
Input Variable	Type	Corresponding data
<code>_MerchantId</code>	Integer	Merchant identifier
Return value : string <code>&lt;id&gt;:&lt;label&gt;:&lt;prefs&gt;;&lt;id&gt;:&lt;label&gt;:&lt;prefs&gt;...</code>		

#### 2.6.1.3.1 Example of use

This call

```

«VAR NEW _UserGroupList=""»
«INCLUDE
_UserGroupList=ossbo:OSSUserGroupList.phs;_MerchantId=«_MerchantId»»

```

stores into `_UserGroupList`, as an example, "5:Accounting:00000000;1:Administrator:10000000;...". The `<prefs>` string consists in a 8-digit boolean string coding for user group preferences ; as of now, only the first digit is used meaning if group is open (0) or closed (1).

#### 2.6.1.4 Information on user groups

Calling the script `ossbo:OSSUserGroupInfo.phs` allows to obtain the label, preferences and default user identifier of user group as identified by `_UserGroupId`.

ossbo:OSSUserGroupInfo.phs		
Input Variable	Type	Corresponding data
<code>_MerchantId</code>	Integer	Merchant identifier
<code>_UserGroupId</code>	Integer	User group identifier
<code>_DefaultUserId</code>	Integer	Default user identifier
Output Variable	Type	Corresponding data
<code>_UserGroup</code>	String	User group label
<code>_UserGroupPrefs</code>	String	User group preferences
Return value : <code>_Status</code> (1 if group was found, 0 if not)		

##### 2.6.1.4.1 Example of use

For instance :

```

«VAR NEW _UserGroup=""»
«VAR NEW _UserGroupPrefs=""»
«VAR NEW _DefaultUserId=0»
«INCLUDE
ossbo:OSSUserGroupInfo.phs;_MerchantId=«_MerchantId»;_UserGroupId=5»

```

would store the name of the user group as well as its preferences (see *Getting list of user groups, Example of use* above) in the initialized variables for user group number 5.

## 2.6.2 Information on Merchant

The data entered at the *Merchant General Information Page* can be retrieved through the `ossbo:OSSMerchantInfo.phs` script.

ossbo:OSSMerchantInfo.phs		
Input Variable	Type	Corresponding data
_MerchantId	Integer	Merchant identifier
_LanguageId	Integer	Language identifier (2.3.3) [Optional]
Output Variable	Type	Corresponding data
_MerchantName	String	Merchant name
_MerchantAddress1	String	Merchant address
_MerchantAddress2	String	
_MerchantAddress3	String	
_MerchantPCode	String	Merchant postcode
_MerchantCity	String	Merchant city
_MerchantCountryId	Integer	Merchant country identifier
_MerchantCountry	String	Merchant country name
_MerchantPhone	String	Merchant phone number
_MerchantFax	String	Merchant fax number
_MerchantEmail	String	Merchant email
_MerchantPrefs	String	Merchant preferences string
_MerchantUserGroupId	Integer	Merchant user group identifier
Return value : <code>_Status</code> (1 if merchant info was correctly fetched, 0 if not)		

### 2.6.2.1 Example of use

An example on the purpose of recuperating some data about your merchant is cleared up at *Advanced developing : using Blue Chameleon's mail gate (2.9)*.

### 2.6.3 Information on clients

The script `ossbo:OSSClientInfo.phs` aims at getting all client information as entered/modified at the *Add/Modify Client Page*.

ossbo:OSSClientInfo.phs		
Input Variable	Type	Corresponding data
_MerchantId	Integer	Merchant identifier
_LanguageId	Integer	Language identifier (2.3.3) [Optional]
_ClientId	Integer	Client identifier
_ClientUsername	String	Client username [Optional]
_NoWalletInfo	Integer	Flag to ignore wallet (0-1) [Optional]
Output Variable	Type	Corresponding data
_ClientType	Integer	Client type
_ClientTitle	String	Client title
_ClientCompany	String	Client company name
_ClientFName	String	Client first name
_ClientName	String	Client name
_ClientFullName	String	Client full name (name, first name, company)
_ClientAddress1	String	Client address
_ClientAddress2	String	
_ClientAddress3	String	
_ClientPCode	String	Client postcode
_ClientCity	String	Client city
_ClientCountryId	Integer	Client country identifier
_ClientCountry	String	Client country
_ClientPhone	String	Client phone number
_ClientOffice	String	Client office phone number
_ClientMobile	String	Client mobile phone number
_ClientFax	String	Client fax number
_ClientEmail	String	Client email address
_ClientUsername	String	Client username
_ClientPassword	String	Client password
_ClientLanguage	String	Client language
_ClientDelivery	String	Client delivery
_ClientDeliveryData	String	Client delivery data
_ClientBilling	String	Client billing
_ClientPayment	String	Client payment
_ClientPaymentData	String	Client payment data
_ClientPaymentStatus	String	Client payment status label
_ClientSLA	String	Client service level agreement
Return value : <code>_Status</code> (1 if client was found, 0 if not)		

### 2.6.3.1 Example of use

The following displays a table of clients' full names and phone numbers :

```
<TABLE>
<TR>
<TD>Name</TD><TD>Number</TD>
</TR>
```

```
«SQLREPEAT»
«VAR NEW _ClientName=""»
«VAR NEW _ClientPhone=""»
«INCLUDE _Status=ossbo:OSSClientInfo.phs;_MerchantId=1;_ClientId=«#SQLREPEAT#»»
«IF _Status==1»
<TR>
  <TD>«_ClientName»</TD><TD>«_ClientPhone»</TD>
</TR>
«ENDIF»
«/SQLREPEAT 0;1000»
</TABLE>
```

## 2.6.4 Information on subscriptions

The script `ossbo:OSSSubscriptionInfo.phs` returns various information on subscriptions, and also on the related client.

ossbo:OSSSubscriptionInfo.phs		
Input Variable	Type	Corresponding data
_MerchantId	Integer	Merchant identifier
_LanguageId	Integer	Language identifier (2.3.3) [Optional]
_SubscriptionId	Integer	Subscription identifier
Output Variable	Type	Corresponding data
_SubscriptionParentId	Integer	Parent subscription identifier
_SubscriptionChildList	String	List of subscription children identifiers
_SubscriptionPrevId	Integer	Previous subscription identifier
_SubscriptionNextId	Integer	Next subscription identifier
_SubscriptionName	String	Subscription name
_SubscriptionStatus	String	Subscription status (<iStatus>=<label>)
_SubscriptionCustomStatus	String	Subscription custom status (<id>=<label>)
_SubscriptionStartDate	Integer	Subscription start date (cdate)
_SubscriptionEndDate	Integer	Subscription end date (cdate)
_ArtId	Integer	Article identifier
_ArtPrice	Integer	Article price
_ArtBasePrice	Integer	Article base price
_ArtFollowUpPrice	Integer	Article follow-up price
_ArtUnits	Integer	Article units
_ArtUnitsType	Integer	Article units type
_ArtUnitsPrice	Integer	Article units price
_ArtLibraryData	String	Article external library data
_OrderNumber	String	Order reference number
_OrderDate	Integer	Order date as ctime
_OrderSalesman	String	Order salesman identifier
_OrderPayment	String	Order payment method
_OrderBilling	String	Order billing method
_OrderTicketId	Integer	Order ticket identifier
_ClientId	String	Client identifier (Param=<id>)
⊕ All ossbo:OSSClientInfo.phs (2.6.3) output variables		
Return value : <b>_Status</b> (1 if subscription was found, 0 if not)		

#### 2.6.4.1 Example of use

Part *Developing subscription (configure, display) custom scripts : an example* (2.7.2.2) shows in detail how `ossbo:OSSSubscriptionInfo.phs` can be useful.

#### 2.6.5 Information on orders

The script `ossbo:OSSOrderInfo.phs` returns various information on orders, and also on the related client.

ossbo:OSSOrderInfo.phs		
Input Variable	Type	Corresponding data
_MerchantId	Integer	Merchant identifier
_LanguageId	Integer	Language identifier (2.3.3) [Optional]
_OrderId	Integer	Order identifier
Output Variable	Type	Corresponding data
_OrderOwner	Integer	Order owner identifier
_OrderNumber	String	Order reference number
_OrderDate	Integer	Order date as ctime
_OrderSalesman	String	Order salesperson identifier
_OrderPayment	Integer	Order payment method
_OrderBilling	Integer	Order billing method
⊕ All ossbo:OSSClientInfo.phs (2.6.3) output variables		
Return value : _Status (1 if client and order were found, 0 if not)		

### 2.6.5.1 Example of use

The following outputs a table of order numbers, salespersons and client names of orders done in 2010.

```

<TABLE>
<TR>
  <TD>Order</TD><TD>Salesperson<TD>Client</TD>
</TR>
<SQLREPEAT>
  <VAR NEW _OrderNumber="">
  <VAR NEW _OrderSalesman="">
  <VAR NEW _ClientName="">
  <INCLUDE ossbo:OSSOrderInfo.phs;_MerchantId=1;_OrderId=<#SQLREPEAT#>>
  <IF @strstrsearch(_OrderNumber;"2010")==0>
  <TR>
    <TD><_OrderNumber></TD><TD><_OrderSalesman></TD><TD><_ClientName></TD>
  </TR>
  <ENDIF>
</SQLREPEAT 0;10000>
</TABLE>

```

## 2.6.6 Information on invoices

The script ossbo:OSSBillInfo.phs returns various (mostly numerical) information on invoices.



ossbo:OSSBillInfo.phs		
Input Variable	Type	Corresponding data
_MerchantId	Integer	Merchant identifier
_BillId	Integer	Invoice identifier
Output Variable	Type	Corresponding data
_BillOrderId	Integer	Invoice order identifier
_BillUserId	Integer	Invoice user identifier that registered the invoice
_BillClientId	Integer	Invoice client identifier
_BillReference	String	Invoice reference
_BillCurrencyId	Integer	Invoice currency identifier
_BillDate	Integer	Invoice date (cday)
_BillValueDate	Integer	Invoice value date (cday)
_BillTotalPrice	Integer	Invoice total price
_BillTotalTax	Integer	Invoice total tax
_BillOpenPrice	Integer	Invoice total open price
_BillStatus	Integer	Invoice status (0=editable, 1=sent, 2=reminder 1, 3=reminder 2, 4=reminder 3, 5=reminder 4, 100=finished)
_BillPaid	Integer	Invoice paid status (0=unpaid, 1=paid)
_BillExport	Integer	Invoice export status (0=not exported, 1=exported, 2=exported and confirmed)
Return value : <b>_Status</b> (1 if invoice was found, 0 if not)		

### 2.6.6.1 Example of use

The following shows a table featuring already-paid invoices over a certain amount (**\_MyAmount**) as recorded in 2010, giving also the user's names that recorded them :

```

<TABLE>
<TR>
<TD>Reference</TD><TD>Amount</TD><TD>Recorded by</TD>
</TR>
<SQLREPEAT>
<VAR NEW _BillReference="">
<VAR NEW _BillTotalPrice=0>
<VAR NEW _BillUserId=0>
<VAR NEW _BillPaid=0>
<INCLUDE _Status=ossbo:OSSBillInfo.phs;_MerchantId=1;_BillId=<#SQLREPEAT#>>
<IF _Status==1 && _BillPaid==1 && _BillTotalPrice>_MyAmount>
<VAR NEW _UserRealname="">
<INCLUDE ossbo:OSSUserInfo.phs;_MerchantId=<_MerchantId>;_UserId=<_BillUserId>
<TR>
<TD><_BillReference></TD><TD><_BillTotalPrice></TD><TD><_UserRealname></TD>
</TR>
<ENDIF>
</SQLREPEAT 0;10000>

```

</TABLE>

## 2.7 Advanced developing : custom scripts

The previous showed how information related to Blue Chameleon objects could be retrieved. Now, conversely, when dealing with articles, clients, subscriptions... in the regular shop, you might want to, *in this very environment*, link to data pertaining to your add-on.

This can be achieved thanks to *custom scripts*, where a particular, dedicated script belonging to your add-on is declared in the Shop environment to be actionable there. For instance, while on a subscription search, buttons for each subscription may be available to display further data.

### 2.7.1 Adding a custom script



*User rights for custom script management can be set up either at the Modify User Page or the User Right Page under the element*

#### Script management

No matter what its context use will be, a custom script is created as shown in Fig.2.11.

General / Personalization / Custom scripts / Add new custom script

**Add custom script**

Script: "Display subscription" Script

Library: AddOn - AddOn

Name: View License

Button text: View licence;Voir license;Lizenz anzeigen

Script file: AddOn:ViewLicence.phs

Add

Figure 2.11: The *Add New Menu Script Page*.

On this page, the following are chosen :

- a use, through the 'Script:' menu ;
- the library it belongs to (i.e. your add-on) ;
- a name under which it appears on the page where this script is selected ;

- the text of the button that will launch it ;
- finally, the script file itself, always preceded by the add-on name.

Once created, the rights for the custom script must be set on the *Modify User Page* (Fig.2.12).

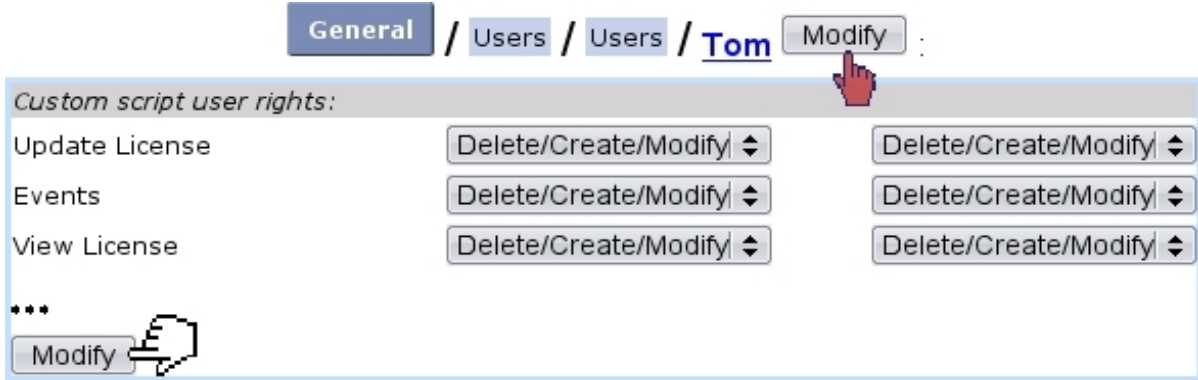


Figure 2.12: All defined custom scripts are assessed rights so that the relevant buttons (if applicable) might be displayed.

## 2.7.2 Subscription-related custom scripts

They are of different types, as chosen from the 'Script:' menu on the *Add New Menu Script Page* as illustrated above :

- "Configure subscription" ;
- "Display subscription" ;
- "Display subscription consumption" ;
- "Event on subscription".

The first three aim at displaying relevant buttons for each subscription after a subscription search (through either **Existing subscriptions** or **Search subscriptions**), while the latter aims at triggering dedicated actions when subscription status is modified (activated, suspended,...).

### 2.7.2.1 Attaching scripts to a subscription article - Effects

For a particular subscription article, defining (if desired) which custom scripts are triggered is done while at the *Modify Subscription Page* (Fig.2.13), accessed through the eponymous button as found on the *Article Management Page* (see *Article Management* chapter, *Managing articles*).

The 'Configuration script', 'View script', 'Consumption script' and 'Event script' menus respectively feature defined custom scripts of the types as listed above. Concerning the 'Event script', the checkboxes respectively rule :

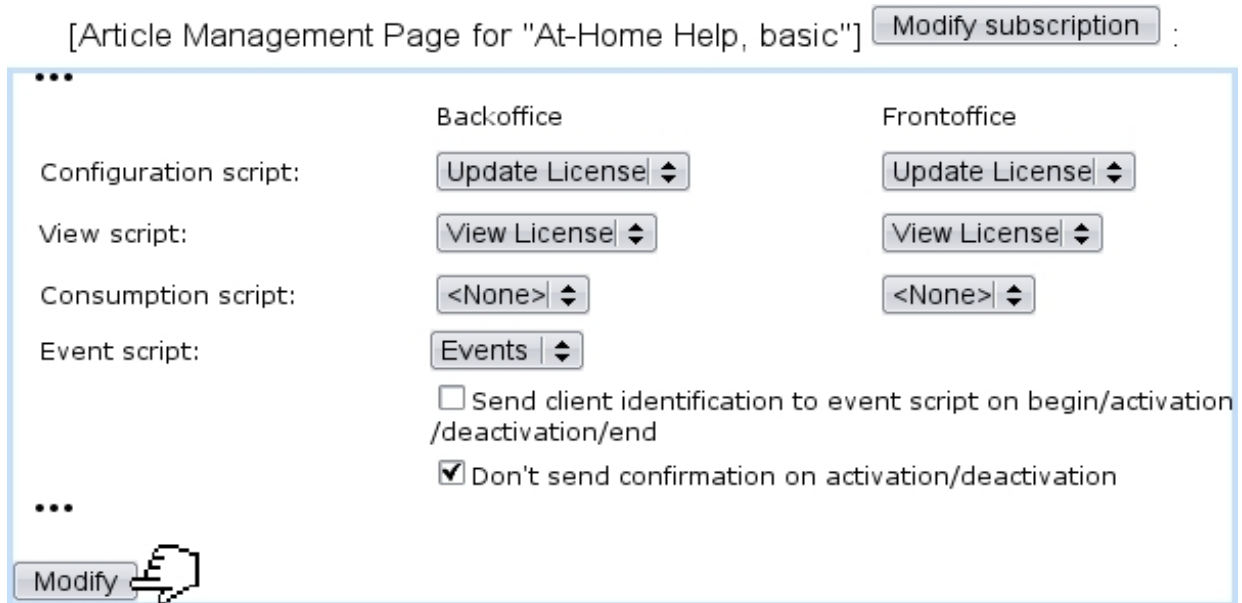


Figure 2.13: Picking previously-defined subscription custom scripts to be executed on this subscription article.

- if variables `_OssUserName` and `_OssUserPassword` (related client's user name and password) will be available in the 'Event script' ;
- whether an email is automatically sent to client on activating or de-activating. In the former case (with email sent), variable `_iCommand` (see table below) will be respectively 7 or 8, while in the latter 5 or 6.

Fig.2.14 then shows the new buttons that appear for a particular subscription which article was assessed "Configure subscription" and "Display subscription" scripts.

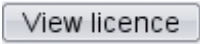


Figure 2.14: These buttons now appear for any "At-Home Help" subscription.

Then, in the scripts as called by the buttons or triggered by a subscription status change, several variables exist :

Custom script type	Existing variable	Corresponding data
All	<code>_MerchantId</code> <code>_CustomScriptId</code>	Merchant identifier Custom script identifier
Configure subscription Display subscription Display subscription consumption	<code>_SubscriptionId</code>	Subscription identifier
Event on subscription	<code>_iOssIdentifier</code>  <code>_iCommand</code>   <code>_iStartDate</code> <code>_iEndDate</code>  <code>_OssUserName</code> <code>_OssPassword</code>	Subscription identifier  New subscription status : 1=Started 2=Ended 5=Activated (or re-activated) 6=Halted 7=Activated (or re-activated) - with email sent 8=Halted - with email sent 9=Subscription dates modified : - new start date - new end date 10=Consumption updated  Client user name Client password

### 2.7.2.2 Developing subscription custom scripts : "Display" example

The following codes show an example of what can be done for a "Display subscription" custom script named `AddOn:ViewLicense.phs`. Inside the add-on, a table `ADDON_LICENCE` had been previously created with the fields `idSubscription (int)`, `iEndDate (int)` and `azLicense varchar(128)`. This code basically displays these three values for the clicked-on (via button ) subscription.

In Code 13, first, custom script user rights related to this script are loaded and further access is blocked if user does not have at least 'View' user rights (cf. Fig2.12). Then, the correct variables to store the subscription end date and client name to are initialized and set by a call to script `ossbo:OSSSubscriptionInfo.phs` (2.6.4).

Finally, last part of `AddOn:ViewLicense.phs` (Code 14) fetches the "license" corresponding to the subscription and displays a table where subscription identifier, end date and license are shown. Concerning end date, a small test is done to check if subscription has no specified end (in which case Blue Chameleon had stored corresponding `cdate` as 99999), otherwise formats it in DD/MM/YYYY.

Fig.2.15 shows this example "Display subscription" script's final result.

For this page to actually display information, the `ADDON_LICENCE` line as used here

---

### Code 13 AddOn:ViewLicense.phs : rights, initializing variables

---

```
«VAR NEW _OwnerUserRights=""»
«VAR NEW _UserRights=""»
«INCLUDE
ossbo:OSSCustomUserRights.phs;_MerchantId=«_MerchantId»;_CustomScriptId=«_CustomScriptId»»

«*Is user allowed to do this ?*»
«IF _UserRights[0]<49»
  «INCLUDE ossbo:MessageAccessDenied.phs#Message»
  «EXIT STOP»
«ENDIF»

«VAR NEW _SubscriptionEndDate=0»
«VAR NEW _ClientName=""»
«INCLUDE
ossbo:OSSSubscriptionInfo.phs;_MerchantId=«_MerchantId»;_SubscriptionId=«_SubscriptionId»»
```

---

---

### Code 14 AddOn:ViewLicense.phs [continued and finished] : display of results

---

```
«SQLEXEC STRING _License=SELECT azLicense FROM ADDON_LICENSE WHERE
idSubscription=«_SubscriptionId»»

<p>
<table>
  <tr><td>Subscription identifier :</td><td>«_SubscriptionId»</td></tr>
  <tr><td>Client :</td><td>«_ClientName»</td></tr>
  <tr>
    <td>End date :</td>
    «IF _SubscriptionEndDate=99999»«*not specified*»
    <td>Not specified</td>
    «ELSE»
    <td>«=@daytimeday(_SubscriptionEndDate)»/
      «=@daytimemonth(_SubscriptionEndDate)»/
      «=@daytimeyear(_SubscriptionEndDate)»</td>
    «ENDIF»
  </tr>
  <tr><td>License :</td><td><b>«_License»</b></td></tr></table>
<p>
```

---



Figure 2.15: The rendering of `AddOn:ViewLicense.phs`. It is to remember that the title of the page is not set in the script itself, but by the 'Name:' as entered while adding this custom script (Fig.2.11).

had to be inserted in the first place : this can be done automatically by using a "Event on subscription" custom script, as cleared up below.

### 2.7.2.3 Subscription events : an example

An "Event on subscription" script, or more precisely one of its parts, is automatically launched when a particular subscription is activated, suspended, modified date-wise,...

The general structure of such a custom script, called in this example `AddOn:Events.phs` is better done following the guideline as shown in Code 15. Basically, already-existing variable `_iCommand` is tested so that specific commands (or even a call to an add-on script) are performed according to how subscription has been acted on.

In the example of the `ADDON_LICENCE` table, the `AddOn:Events.phs` can be equipped with commands that insert a new line in this table when subscription is activated.

In the SQL transaction as featured in Code 16, a test is done on whether this subscription is already in the `ADDON_LICENSE` table (so that it may not be inserted another time in the case of a re-activating) ; if not :

- a "license" string is composed by concatenating client's name (put into capitals and simplified thanks to `@tocompare` function) with a random number, casted into a string ;
- this element is inserted into the table along with subscription identifier and end date.

It is to note that in this "Event on subscription" context, subscription is identifier by `_iOssIdentifier`.

At the end, variable `_Success` records the success or failure of the transaction, and result is displayed on the screen, as featured in Fig.2.16.

---

**Code 15 AddOn:Events.phs** : a general template.

---

```
«VAR NEW _OwnerUserRights=""»
«VAR NEW _UserRights=""»
«INCLUDE
ossbo:OSSCustomUserRights.phs;_MerchantId=<_MerchantId>;_CustomScriptId=<_CustomScriptId>»

«IF _iCommand==1»
  [Do stuff associated with this subscription's start]
«ELSEIF _iCommand==2»
  [Do stuff associated with this subscription's end]
«ELSEIF _iCommand==5»
  [Do stuff associated with this subscription's activating or re-activating]
«ELSEIF _iCommand==6»
  [Do stuff associated with this subscription's suspension or block]
«ELSEIF _iCommand==7»
  [Do stuff associated with this subscription's activation or re-activation and send
email]
«ELSEIF _iCommand==8»
  [Do stuff associated with this subscription's suspension or block and send email]
«ELSEIF _iCommand==9»
  [Do stuff when this subscription's start and end dates are modified]
«ELSEIF _iCommand==10»
  [Do stuff when this subscription's consumption is updated]
«ENDIF»
```

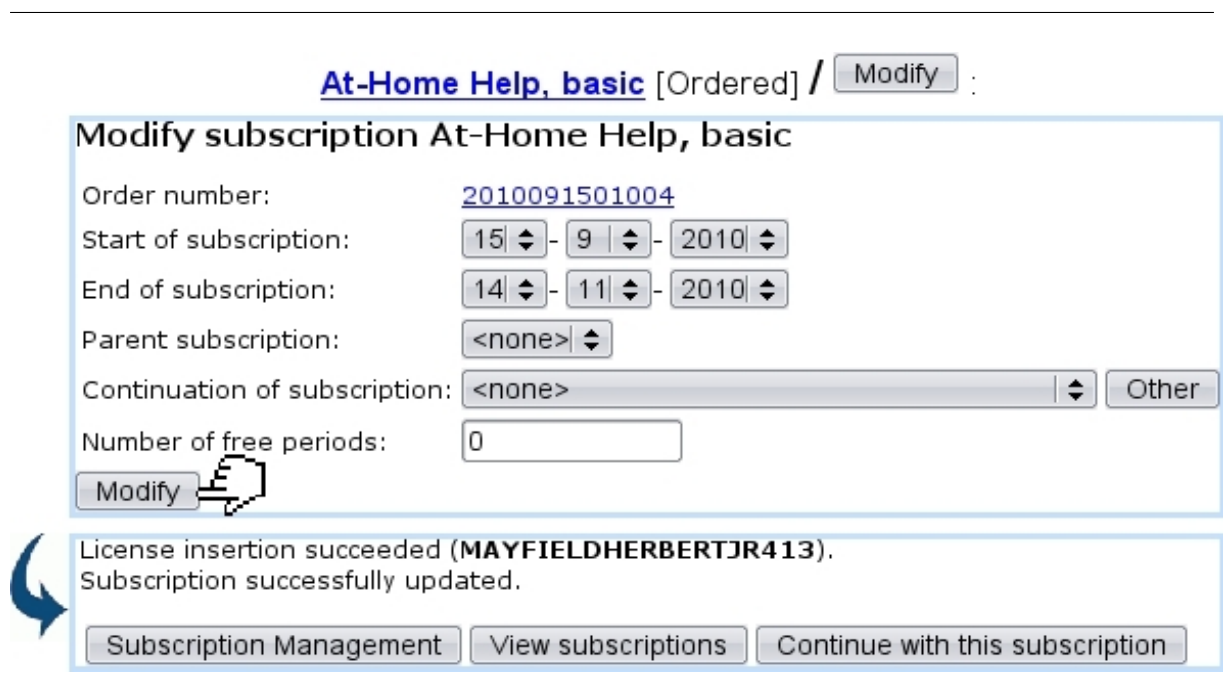


Figure 2.16: This ordered subscription is activated, which triggers the corresponding event in AddOn:Events.phs.

The following code shows another example : it aims at updating ADDON\_LICENSE where subscription dates are modified.



---

**Code 16** AddOn:Events.phs [part] : do this when subscription is activated

---

```
...
<VAR NEW _SubscriptionEndDate=0>
<VAR NEW _ClientName="">
<INCLUDE
ossbo:OSSSubscriptionInfo.phs;_MerchantId=<_MerchantId>;_SubscriptionId=<_iOssIdentifier>>

<IF _iCommand==5 || _iCommand==7 >
  <SQLSTATUS TRANSACTION BEGIN>
  <SQLEXEC INT _Exists=SELECT COUNT(*) FROM ADDON_LICENSE WHERE
idSubscription=<_iOssIdentifier>>
  <IF _Exists==0>
    <VAR NEW _License=@tocompare(_ClientName)+@ittoa(@random(1000))>
    <SQL INSERT INTO ADDON_LICENSE
VALUES(<_iOssIdentifier>,<_SubscriptionEndDate>,'<_License>')>
  <ENDIF>
  <VAR NEW _Success=#TRANSACTIONSTATUS#>
  <SQLSTATUS TRANSACTION END>
  <IF _Exists==0>
    <IF _Success==1>
      License insertion succeeded (<b><_License></b>).
    <ELSE>
      License insertion failed !
    <ENDIF>
  <ENDIF>
<ENDIF>
...

```

---

---

**Code 17** AddOn:Events.phs [part] : do this when subscription's dates are modified

---

```
...
<IF _iCommand==9 >
  <SQL UPDATE ADDON_LICENSE SET iEndDate=<_iEndDate> WHERE
idSubscription=<_iOssIdentifier>>
  <IF #SQLSTATUS#==1>
    License entry successfully updated.
  <ELSE>
    License entry update failed !
  <ENDIF>
<ENDIF>
...

```

---

The new end date for that subscription is recuperated through the `_iEndDate` variable and success is shown on the final screen (Fig.2.17).

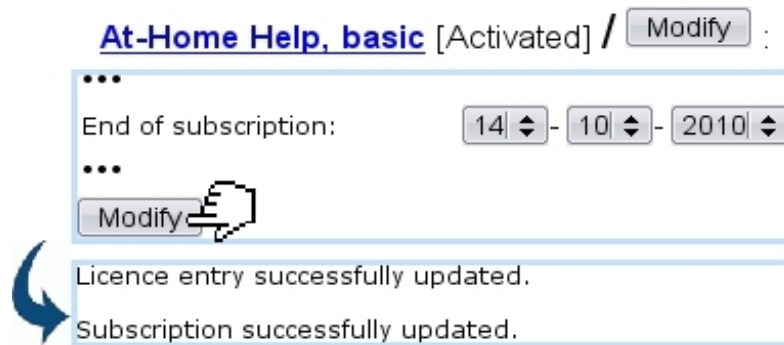


Figure 2.17: This activated subscription has its end date modified, which triggers the corresponding event in `AddOn:Events.phs`.

## 2.8 Advanced developing : a custom event control

Blue Chameleon is naturally equipped with event controls (see *General Interface Handling, Event Controls*) that, when a user logs on, aim to represent the latest tickets, the current/late projects, tasks...

Your add-on can also be featured on Blue Chameleon's **Home**. In order to do this, an `oss-` script (2.3.1) called `ossModuleHeadline.phs` can be developed.

Code 18 shows an example. The code for your headline must be embedded in the `ELSE...ENDIF` environment, as the first part enables Blue Chameleon to propose your event control to be added, through **Options** / **Configure desktop** /  **AddOn** **Add**.

After the mandatory `IF... preamble`, the example shows how custom user rights are recuperated :

- first, a call to `ossbo:OSSCustomScriptId.phs` is done to recall the identifier of the custom script for your add-on ; '2001' must be used for the `_CustomScript` parameter (it identifies it as 'menu script') and `_RegisterId` is no other than the signature of your add-on (2.3.1.1), here 1003 ;
- the fetched `_CustomScriptId` then enables to recuperate the custom script user rights, through a call to `ossbo:OSSCustomUserRights.phs`.

If needed, the custom rights related to the library can be recuperated as shown in Code 12.

In the rest, the `c-date` for the current day is evaluated and used to count how many "licenses" end on this day ; if there actually are, through a `SQLOUTPUT` loop, a maximum

of four of them are then displayed as click-links to a `AddOn:ViewAFinLicense.phs` script to which the following values are transmitted :

- the shop identifier `<SqlDB>`, session identifier `<xid>`, aimed `AddOn:ViewAFinLicense.phs` script ;
- merchant identifier `<_MerchantId>`, custom script identifier `<_CustomScriptId>` ;
- finally, the identifier for that particular subscription `<idSubscription>`.

At the end, another click-link redirects to a script `AddOn:ViewAllFinLics.phs` proposing to view all finishing licenses of the day.

---

**Code 18** `ossModuleHeadline.phs` : the dark-blue part is mandatory.

---

```
<IF @toi(_ossProcInfo)==1>
  <VAR NEW azProcInfo = "AddOn">
  <RETURN azProcInfo>
<ELSE>

  <INCLUDE _CustomScriptId=ossbo:OSSCustomScriptId.phs;_CustomScript=2001;
_MerchantId=<_MerchantId>;_RegisterId=1003>
  <VAR NEW _OwnerUserRights="">
  <VAR NEW _UserRights="">
  <INCLUDE ossbo:OSSCustomUserRights.phs; _MerchantId=<_MerchantId>;
_CustomScriptId=<_CustomScriptId>>

  <VAR NEW _Today=@daytime()>
  <SQLEXEC INT _Count=SELECT COUNT(*) FROM ADDON_LICENSE WHERE iEndDate=<_Today>>
  <IF _Count==0>
    No license ends today
  <ELSE>
    <_Count> license(s) end(s) today :<br>
    <SQLOUTPUT>
    <a href='/Scripts/sql.exe?SqlDB=<SqlDB>&xid=<xid>&Sql=AddOn:ViewAFinLicense.phs
&_MerchantId=<_MerchantId>&_CustomScriptId=<_CustomScriptId>
&_SubscriptionId=<idSubscription>'>
      <azLicense>
      </a>
      <br>
    </SQLOUTPUT SELECT * FROM ADDON_LICENSE WHERE iEndDate=<_Today>;0;4>
    <a href='/Scripts/sql.exe?SqlDB=<SqlDB>&xid=<xid>&Sql=AddOn:ViewAllFinLics.phs
&_MerchantId=<_MerchantId>&_CustomScriptId=<_CustomScriptId>'>
      View all
    </a>
  <ENDIF>

<ENDIF>
```

---

The final result is shown at Fig.2.18.



Figure 2.18: The custom headline resulting from Code 18.

### 2.8.0.1 Content of a click-link

The following code (19) shows how the linked script `AddOn:ViewAFinLicense.phs` could be implemented : in fact, via an `INCLUDE` it uses again the `AddOn:ViewLicense.phs` one (2.7.2.2), giving it the all parameters it needs (merchant, custom script and subscription identifiers). Also, `ossbo:Header.phs` (2.3.2) is used so as to give to this page the same style as others.

---

**Code 19** `AddOn:ViewAFinLicense.phs` : an example of a linked script from an event control.

---

```

«INCLUDE ossbo:Header.phs;_Title="View finishing license";_Style="print";»

<H1>View finishing license</H1>

  «INCLUDE AddOn:ViewLicense.phs;_MerchantId=«_MerchantId»;
  _CustomScriptId=«_CustomScriptId»;_SubscriptionId=«_SubscriptionId»»

<P>
<input type="button" value="Send mail invite"
onClick="window.location.href='/Scripts/sql.exe?SqlDB=«SqlDB»&xid=«xid»
&Sql=AddOn:InviteMailSend.phs&_MerchantId=«_MerchantId»
&_CustomScriptId=«_CustomScriptId»&_SubscriptionId=«idSubscription»'»

```

---

The output of this is featured at Fig.2.19.

The button at the end redirects to a `AddOn:InviteMailSend.phs` script aimed at sending an automated email to the client whose subscription is finished, inviting her/him to start it again : this uses Blue Chameleon email features and is going to be explained below.



Figure 2.19: A finished "license", as accessed directly from the event control.

## 2.9 Advanced developing : using Blue Chameleon's mail gate

While Blue Chameleon offers a wide palette for confirmation, invoice-reminding,... emails, you nonetheless might want a specific kind of email to be sent, as requested by your add-on's features.

For instance, in the add-on example that has been used throughout this chapter, one might want, by a single click, to generate and send an email to a client whose particular subscription has just ended in order to propose her/him to renew it. Code 20 then shows how it could be done, through a `AddOn:InviteMailSend.phs` script.

First, calls to information scripts enable to :

- `ossbo:OSSSubscriptionInfo.phs` : get the name of the subscription article and the order number related to this finished subscription ; client identifier too, under the form `Param=5` (for instance), which needs a string extract and cast to be done ;
- `ossbo:OSSClientInfo.phs` : get client's email, title (Mr, Mrs,...) and last name ;
- `ossbo:OSSMerchantInfo.phs` : get your Merchant's email and name.

Then, mail-related commands are called, first to declare sender as your merchant email (`MAIL FROM`), recipient as the client email (`MAIL TO`) and a mail subject featuring the name of the subscription article (`MAIL SUBJECT`).

Next, the mail's text itself begins, between a `OUTPUT MAIL.../OUTPUT` environment, in which `OSL` commands can be used. A `MAIL SEND` finally sends this text to the client's email and user is informed of the success of the sending.

---

**Code 20** AddOn:InviteMailSend.phs This script sends a dedicated email to a single client.

---

```
«*Subscription identifier _SubscriptionId and _MerchantId must be inherited from
calling script !*»

«VAR NEW _ClientId=""»
«VAR NEW _SubscriptionName=""»
«VAR NEW _OrderNumber=""»
«INCLUDE
ossbo:OSSSubscriptionInfo.phs;_MerchantId=«_MerchantId»;_SubscriptionId=«_SubscriptionId»»

«LET _ClientId=@toi(@substr(_ClientId;6;-1))»

«VAR NEW _ClientEmail=""»
«VAR NEW _ClientTitle=""»
«VAR NEW _ClientName=""»
«INCLUDE ossbo:OSSClientInfo.phs;_MerchantId=«_MerchantId»;_ClientId=«_ClientId»»

«VAR NEW _MerchantEmail=""»
«VAR NEW _MerchantName=""»
«INCLUDE ossbo:OSSMerchantInfo.phs;_MerchantId=«_MerchantId»»

«MAIL FROM «_MerchantEmail»»
«MAIL TO «_ClientEmail»»
«MAIL SUBJECT About your «_SubscriptionName» subscription...»

«OUTPUT MAIL»
  Dear «IF _ClientTitle<>""»«_ClientTitle»«ENDIF» «_ClientName»,

  Your subscription «_SubscriptionName» (order Nr «_OrderNumber») has just expired
today !

  If you are satisfied with our services, we invite you to renew it.

  Kind Regards,

  The «_MerchantName» Team
«/OUTPUT»

«MAIL SEND»

«IF #SQLSTATUS#==1»
  Email successfully sent to «_ClientEmail»».
«ELSE»
  ERROR : Could NOT send mail to «_ClientEmail»».
«ENDIF»
```

---

## 2.10 Communication add-on to add-on, and shop to add-on : the oss- scripts

Blue Chameleon's great force is that it allows one add-on to perform actions (such as viewing/searching contents, adding an element,...) on an other add-on. Also, when some actions are done in the Shop, some scripts belonging to add-ons, if defined, can be triggered.

This all relies on the concept of *objects* in an explicit manner (in the case of add-on to add-on) or implicit manner (shop to add-on), An object is defined by a triplet of values :

- the identifier of the library it belongs to, as defined in the add-on's `ossModuleRegister.php` script 2.3.1.1 ;
- an identifier to qualify the type of this object ;
- an identifier for the object itself.

As variables, these values are often called `_LibraryId`, `_ObjectTypeId` and `_ObjectId`. Inside an add-on, this triplet fully identifies an object.

By principle, an add-on cannot directly act on the objects of an other add-on, but instead calls scripts belonging to the Shop, which then serves as an intermediary ; it will then call the relevant script aimed add-on (Fig.2.20). The Shop, on the other hand, can act on any add-on, if the corresponding scripts have been implemented for them. These scripts have standardized names. A Shop's script always starts by `OSS...` ; an add-on script by `oss-`

For instance, if add-on "Alpha" wants add-on "Beta" (registered as 1003) to display a list of its objects Widget (which are for instance type 2), the following call is written in add-on Alpha :

```
«INCLUDE
ossbo:OSSObjectSearch.php;_MerchantId=«_MerchantId»;_ObjectTypeId=2;_LibraryId=1003;
[Other parameters]»
```

The shop script's `OSSObjectSearch.php` will then call add-on Beta's `ossObjectSearchObj.php` script (which has to be developed) with several parameters amongst which the object type.

Several scripts of this kind exist ; their name all start with `oss-`. Once developed, they have to be registered in order to be usable.

### 2.10.1 Registering oss- scripts

As any oss- script is written, a recheck of the library must be done, through [Articles](#) / [Personalization](#) / [Personalization](#) / [Custom libraries](#) / [Library name](#) [Register again](#).

Registered oss- scripts as well as objects available for a add-on (see below) can be checked from the previous link (Fig.2.21).

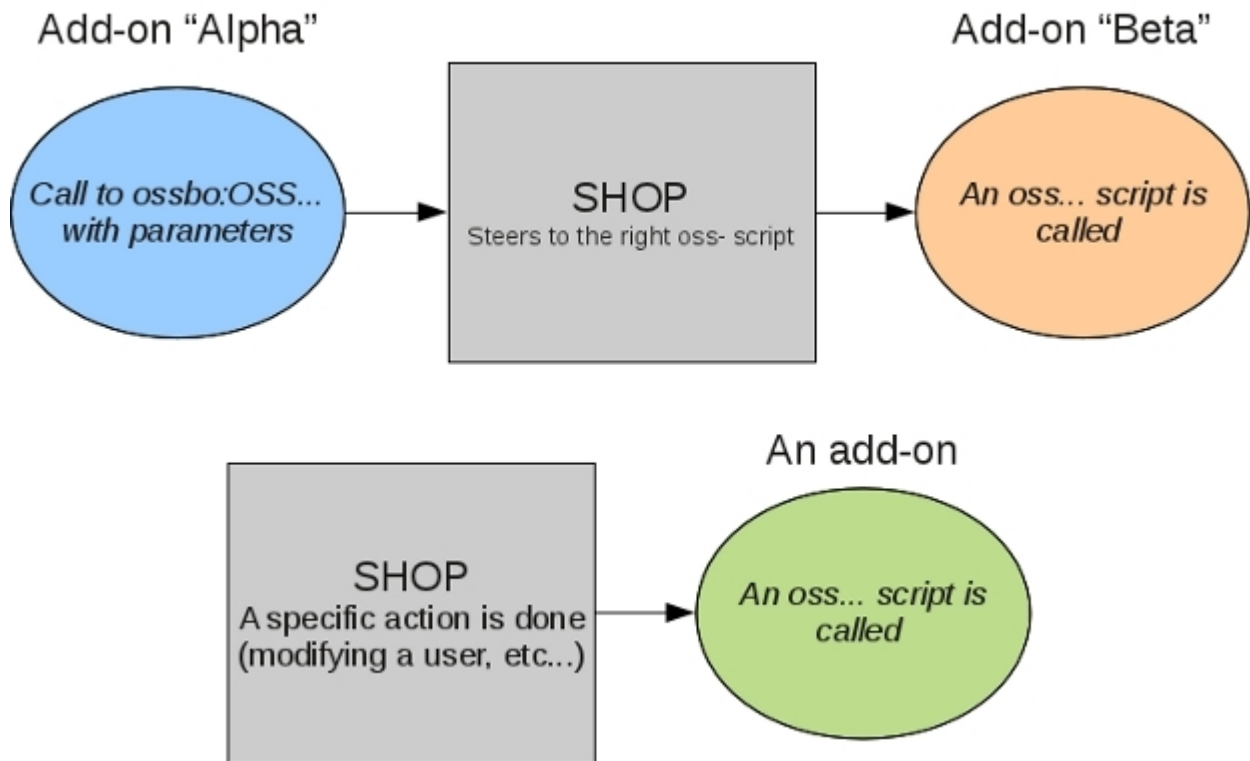


Figure 2.20: The principles of add-on to add-on and shop to add-on interactions.

## 2.10.2 Recording objects inside an add-on

For objects to be handled inside an add-on, they have to be registered first through a `ossModuleInit.phs` script where the types of the various objects are recorded through a Shop's script. Code 21 shows how `ossModuleInit.phs` is made.

---

**Code 21** `ossModuleInit.phs` This records add-on Beta's objects.

---

```

«LET _ObjectName="Gizmo"»
«INCLUDE ossbo:OSSObjectTypeAdd.phs;_ObjectType=1;_ObjectTypePrefs="00"»
«LET _ObjectName="Widget"»
«INCLUDE ossbo:OSSObjectTypeAdd.phs;_ObjectType=2;_ObjectTypePrefs="00"»
  
```

---

There, the add-on's objects `Gizmo` and `Widget` are thus respectively recorded with the types 1 and 2.

Once done, the library must be recompiled and rechecked (2.10.1).

## 2.10.3 Integrating properties in an add-on

In the following example, the add-on in which to include properties has '1003' as identifier (2.3.1.1).



## Beta :

**View custom library**  
Library: Beta  
Signature: 1003  
Label: Beta  
Type: General  
**Registered object types:**  
1 Gizmo  
2 Widget  
**Available scripts:**  
ossModuleInit OK  
ossModuleLogin not implemented  
ossModuleLogout not implemented  
ossModuleMessage not implemented  
ossModuleHeadline OK  
ossModuleObjectType not implemented  
ossModuleLoad not implemented  
ossModuleStat not implemented  
ossObjectStatusSet not implemented  
ossObjectBillLabel not implemented  
ossObjectQueryInfo not implemented  
ossObjectSearchObj OK  
ossObjectAdd not implemented  
ossObjectModify not implemented  
ossObjectDelete not implemented  
ossObjectView not implemented  
...

Figure 2.21: Objects and scripts that an add-on possesses.

### 2.10.3.1 Registering the properties to be used

For instance, we want to use two properties called 'Active' for a client and 'Rank' for a salesman.

The add-on's script `ossModuleInit.phs` (2.10.2) must then contain :

```
«VAR NEW _Name1="Active;Actif;Aktiv"»  
«INCLUDE ossbo:OSSObjectPropAdd.phs;_MerchantId=<idMerchant>;  
_ObjectLibraryId=0;_ObjectType=1;_PropLibraryId=1003;  
_PropId=1;_PropertyName=_Name1»  
«VAR NEW _Name2="Rank;Rank;Rank"»  
«INCLUDE ossbo:OSSObjectPropAdd.phs;_MerchantId=<idMerchant>;  
_ObjectLibraryId=0;_ObjectType=14;_PropLibraryId=1003;  
_PropId=2;_PropertyName=_Name2»
```

There :

- `_ObjectLibraryId` is the identifier of the library of the objects for which to handle properties (here, 0 for Shop) ;
- `_ObjectType` is the type of the objects for which to handle properties (here, 1 for Clients and 14 for Salesmen) ;
- `_PropLibraryId` is the identifier of the add-on, 1003 ;
- `_PropId` is the identifier of the property **inside** the add-on (here, 1 and 2) ;

Once done, the library must be recompiled and rechecked (2.10.1).

### 2.10.3.2 Creating the properties

Now, the properties themselves must be created inside the **Property** add-on (2.22).

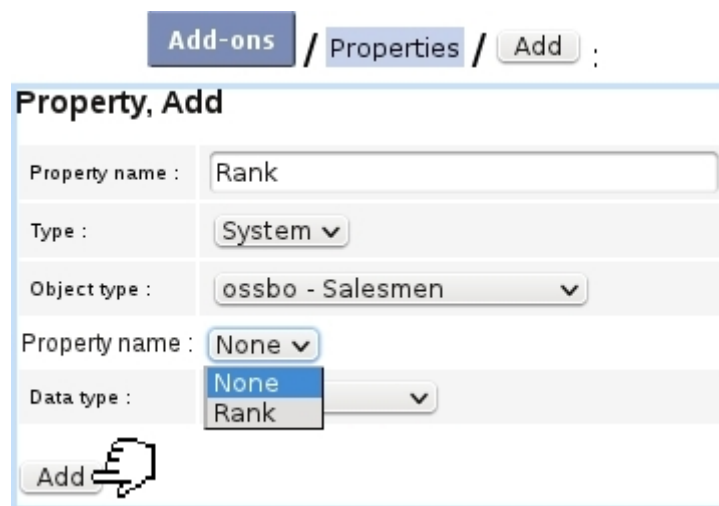


Figure 2.22: Creating the properties.

There, 'Type : System' must be chosen as well as the type of object ; the previously registered property will then appear in the 'Property name' listbox.

That property has then to be configured.

### 2.10.3.3 Using properties inside the add-on

#### 2.10.3.3.1 Getting the list of properties

Inside add-on as identified by 1003, the following call allows to retrieve a string of registered properties along with their data :

```
«VAR NEW _PropObjectList=""»  
«INCLUDE  
_PropObjectList=Property:PropertySystem.phs#GetPropertyObjectList;  
_PropLibraryId=1003»
```

The result string is ... ;<PropId>,<PropertyOwnId>,<LibId>,<ObjId>;... where <PropId> is the property's identifier inside add-on 1003 (i.e. 1, 2), <PropertyOwnId> is the property's identifier inside **Property** add-on (e.g. 27, 28) and <LibId>,<ObjId> are the libraries and object types of the handled objects (i.e 0,1 and 0,14).

So in this example, we would get 1,27,0,1;2,28,0,14;.

The <PropertyOwnId> values are used to set and get property values, so that string has to be obtained and exploited before doing anything.

### 2.10.3.3.2 Creating a property value

If for instance we want to set the value '6' for Property #2, for a (salesman) object `_SomeSalesmanId`, first we get the `_PropObjectList` as shown above ; by cutting the string along ';' and using value '2' as a locator we then extract the property's own identifier, '28'.

The following is then done :

```
«INCLUDE Property:PropertySystem.phs#SetPropertyObjectList;
_PropertyId=28;_LibraryId=0;_ObjectType=14;_ObjectId=«_SomeSalesmanId»;
_Value1=6;_Value2=...;_Value2=...;_Mode=2»
```

This will insert ("\_Mode=2") a property 'Rank' with `_Value1=6` for object (0,14,`_SomeSalesmanId`)

### 2.10.3.3.3 Removing a property value

To remove a property entry for an object identified by (0,14,`_SomeSalesmanId`), the same call as above is made, but using `_Mode=1`.

### 2.10.3.3.4 Setting a property value

If we want to change a property value with a unique occurrence (and what's more, even if we don't know whether it already exists), 'remove' and 'insert' calls can be made successively.

### 2.10.3.3.5 Fetching a property value

In order to retrieve the value of a property for an object identified by (0,14,`_SomeSalesmanId`), the following call is made :

```
«VAR NEW_PropertyValueList=""»
«INCLUDE Property:PropertySystem.phs#GetPropertyObjectValueList;
_PropertyId=28;_LibraryId=0;_ObjectType=14;_ObjectId=«_SomeSalesmanId»»
```

The result string `_PropertyValueList` has the following format : ...@<iOrder>;<iValue1>;<iValue2>;<iValue3>@... There are several chunks separated by '@' if there are multiple entries for that object and property. In that example, we would simply retrieve '@6,0,0,0'.

## 2.10.4 Object management-related scripts

These are add-on to add-on scripts.

Inside an add-on **A**, a shop script is called with parameters, thus calling another add-on **B**'s oss- scripts with certain output parameters.

Action to perform in B	Shop script to be called in A	Result script in B
Adding a new object	ossbo:OSS_ObjectAdd.phs;...	ossObjectAdd.phs
Deleting an object	ossbo:OSS_ObjectDelete.phs;...	ossObjectDelete.phs
Modifying an object	ossbo:OSS_ObjectModify.phs;...	ossObjectModify.phs
Name of an object	ossbo:OSS_ObjectInfo.phs;...	ossObjectQueryInfo.phs
Viewing an object	ossbo:OSS_ObjectView.phs;...	ossObjectView.phs

Input parameters for the `ossbo:OSS...` calls are `_MerchantId`, `_LibraryId` (register id of add-on B), `_ObjectId` (of the object to add, delete, modify, view, ...) and, except for `ossbo:OSS_ObjectAdd.phs;...`, `_ObjectId` (as deleting, modifying, getting the name of or viewing a specific object require it to be fully defined by the triplet).

On add-on B's side, these parameters are then retrieved as they are. The `oss...` scripts in Add-on B should be developed so as to respectively perform the actions of adding, deleting, modifying or viewing and object.

Code 22 shows how, for instance, a `ossObjectAdd.phs` script can be implemented in add-on B.

---

**Code 22** An example of template for a `B:ossObjectAdd.phs` script.

---

```
«IF @toi(_ObjectId)==1 »
  «*perform actions to add an new object of type 1...*»
«ELSEIF @toi(_ObjectId)==2 »
  «*perform actions to add an new object of type 2...*»
...
«ENDIF»
```

---

It is to note that, should an add-on contain several objects, the `«IF @toi(_ObjectId)==1» ... «ELSEIF @toi(_ObjectId)==2» ... «ENDIF»` template must always be used.

### 2.10.4.1 Example : link to view an object, featuring object's name

The following line code (to put in the originating add-on A) outputs a link to the `ossObjectView.phs` of an object of add-on B :

```
<a
href="/Scripts/sql.exe?xid=<xid>&SqlDB=<SqlDB>&_MerchantId=<_MerchantId>
```

```
&_LibraryId=«_idRefObjectLibrary»&_ObjectId=«_idRefObjectLibrary»&_ObjectTypeId=«_idRefObjectType»&_ObjectId=«_idRefObject»&Sql=ossbo:OSS_ObjectView.phs">«_ObjectName»</a>
```

The `_ObjectName` had been recuperated thanks to a call to `ossbo:OSS_ObjectInfo.phs` :

```
[In A :/
«VAR NEW _ObjectName=""»
«INCLUDE
ossbo:OSS_ObjectInfo.phs;_MerchantId=«_MerchantId»&_LibraryId=«_idRefObjectLibrary»
&_ObjectTypeId=«_idRefObjectType»&_ObjectId=«_idRefObject»»
```

```
[In B, : ossObjectQueryInfo]
«SQLEXEC STRING _ObjectName=SELECT azName FROM ... (use _ObjectTypeId,
_ObjectId as query parameters)»
```

## 2.10.5 Searching for an object

### 2.10.5.1 On all libraries' objects

Some add-ons might handle objects from several libraries ; therefore, in order to provide a choice of all available objects, a dedicated Shop script `ossbo:OSSObjectSearch.phs` can be called in the following way in add-on A :

```
«INCLUDE
ossbo:OSSObjectSearch.phs;SqlReturn="A:newObject.phs";_MerchantId=«_MerchantId»;
«CustomScriptId=«_CustomScriptId»»
```

This code produces the following output :



The drop-down menu shows all objects available in all libraries ; choosing a particular object then steers to the corresponding library's `ossObjectSearchObj.phs` script, with, amongst other parameters, the following CGI variables :

- `_SearchObjectId`, a string which value is `<value of chosen library id>:<value of chosen object type>` ; type can be extracted through, for instance, `«VAR NEW _TypeId=@item(2;_SearchObjectId;3) ;`
- `SqlReturn`, which serves as to memorize which script of add-on A to come back to when object in `ossObjectSearchObj.phs` is specifically chosen.

The `ossObjectSearchObj.phs` must provide a form in which objects of type `_TypeId` (extracted as shown above) are displayed, with a submit address equal to `SqlReturn`, the chosen object's triplet being well-defined CGI-wise.

### 2.10.5.2 On a specific library, or specific library's object type

If on the other hand, the library on which to search objects is known, the call to `ossbo:OSSObjectSearch.phs` as shown above has to include a further parameter  `;_LibraryId=...`

Furthermore, if the type of object must be forced, the following must be added :  
`... ;_LibraryId=... ;_ObjectId=...`

### 2.10.6 User-related oss- scripts

The `ossUserConfig.phs` and `ossUserModify.phs` have been respectively detailed at 2.5.3.1 and 2.5.3.2 ; as a reminder, their function is to display an add-on's custom user rights on Shop's *Modify User Page* and to record the modifying thereof.

Other add-on-oriented scripts can be launched when users are managed.

Event in the shop	Script to implement in add-on A	Parameter to use
A new user is added	<code>ossUserAdd.phs</code>	<code>_UserId</code>
A user is deleted	<code>ossUserDelete.phs</code>	<code>_UserId</code>

For instance, if add-on A is equipped with an `ossUserAdd.phs` script, this very script will be triggered each time that a new user is added in the shop. This can be useful for instance if add-on A has a table that requires users to be recorded therein : as a new user is created, `A:ossUserAdd.phs` will be launched, executing for instance `INSERT INTO A_TABLE_USERS VALUES («%:d;_UserId», ...)`, `_UserId` being the shop's inherited variable identifying the new user.

When a user is removed from the shop, an action of deleting a user from some table of add-on A can also be triggered in a similar manner thanks to an implemented `ossUserDelete.phs` script.

### 2.10.7 Article-related oss- scripts

If an article has a defined back-office library (as set on the *Add, Modify Article Page*), the following scripts can be developed in the corresponding library each time a specific action has been done on this article, which is identified as `_ArticleId` in these scripts :

Action	Launched script in back-office library
Article <code>_ArticleId</code> has been added	<code>ossArticleAdd.phs</code>
Article <code>_ArticleId</code> is viewed	<code>ossArticleView.phs</code>
Article <code>_ArticleId</code> 's <i>Modify</i> page is displayed	<code>ossArticleEdit.phs</code>
Article <code>_ArticleId</code> has been modified	<code>ossArticleModify.phs</code>
Article <code>_ArticleId</code> has been deleted	<code>ossArticleDelete.phs</code>

`ossArticleView.phs` and `ossArticleEdit.phs` are used to display view/modify add-on-related information or attributes on the *View/Modify Article* pages ; therefore, those scripts must contain the following template :

```
<TR>
  <TD>
    Some add-on article attribute...
  </TD>
</TR>
<TR>
  <TD>
    Value of that attribute, fetched using _ArticleId, in a View
    or Modify way
  </TD>
</TR>
```

Currently, `ossArticleEdit.phs` is only used at the *Modify Subscription Page* (using an `_EditMode` variable equal to 6) and must follow this template :

```
<<IF @exists("_FirstMod")==1>
  <<IF _EditMode==6>
    (template with TRs, TDs)
  <<ENDIF>
<<ENDIF>
```

## 2.10.8 Client-related oss- script

A `ossClientAction.phs` script can be developed to perform actions in add-on A's tables whenever a client is managed in the shop. Recuperated parameters that are to be used to develop this script are `_ClientId` (identifier of the client that is acted on) and `_Action`, describing what is currently/has been done on the client :

Action	Value of <code>_Action</code>
Client <code>_ClientId</code> has been added	1
Client <code>_ClientId</code> has been modified	2
Client <code>_ClientId</code> has been deleted	3
Client <code>_ClientId</code> 's <i>Modify Page</i> is displayed	5
Client <code>_ClientId</code> is checked for reference (before deletion)	6
Client <code>_ClientId</code> 's <i>Management Page</i> is displayed	7

Generally speaking `ossClientAction.phs` has to follow this template (if add-on requires those actions to be done), using `_ClientId` to identify the client :

```
<<IF _Action==1>
  (for example insert this new client in a table)
<<ELSEIF _Action==2>
  (update some stuff related to that client)
```

```

«ELSEIF _Action==3»
    (delete that client from some table)
«ELSEIF _Action==5»
    (content that will be seen on the Modify Client Page)
«ELSEIF _Action==6»
    (check if client is referenced some table of the add-on ; RETURN
    1 if he isn't, RETURN 0 otherwise)
«ELSEIF _Action==7»
    (content that will be seen on the Client Management Page)
«ENDIF»

```

Values of 5 and 7 allow the add-on to output on the Modify/View Client pages custom content, similarly as the `ossArticleEdit.phs` and `ossArticleView.phs` scripts as used for articles 2.10.7. Their template must then use TRs and TDs.

## 2.10.9 External system-related scripts

The following scripts are launched inside any add-on which has them, they will be triggered when performing actions in the external system and thus will allow to export data pertaining to add-on A similarly as for invoices, payments... The available variable is the identifier of the export `_ExportId` :

Action	Launched script
Count add-on's elements to be exported	<code>ossExportCount.phs</code>
Export add-on's elements to external system	<code>ossExportExternal.phs</code>
Export in add-on is validated	<code>ossExportValidate.phs</code>
Export in add-on is canceled	<code>ossExportCancel.phs</code>

### 2.10.9.1 Examples of implementation

If for instance, in add-on's A, table `PAYMENT_DATA` contains information (references, posting dates, debit/credit amounts) that may be exported to the external system, the `oss-`scripts as mentioned are to be used.

#### 2.10.9.1.1 `ossExportCount.phs`

First, as in the shop, elements to be exported are first counted, this shall also be done for add-on A's data. The following code offers an example of how `ossExportCount.phs` could be implemented : along certain conditions (for instance, of dates), relevant data in `PAYMENT_DATA` is counted and result must be RETURN'ed to be used by the Shop.

```

«VAR NEW _Count=0»
«SQL SELECT COUNT(*) FROM PAYMENT_DATA WHERE <conditions>»
«RETURN _Count»

```



### 2.10.9.1.2 ossExportExternal.phs

Code 23 shows how the template with which this script should be implemented.

There, it can be noted that, for transaction to be exported, some Shop scripts related to exports must be called :

- `ossbo:ProcExport.phs#ExportTransBegin` : start the export of the transaction. Mandatory parameters : `_Reference` of the transaction posting, value date variables for the transaction (`_PostingDay`, `_PostingMonth`, `_PostingYear`, `_ValueDay`, `_ValueMonth`, `_ValueYear`) and an identifier for the user who does it (`_IdUser`) ;
- `ossbo:ProcExport.phs#ExportPostingTag` : write each movement of the transaction. Mandatory parameters : a transaction `_Reference`, an account `_IdAccount`, `_Credit` and `_Debit` values and a `_Label` ;
- `ossbo:ProcExport.phs#ExportTransEnd` : close the transaction. No parameter.

Therefore, the data from the relevant table in the add-on might be extracted so that it respects this framework.

---

**Code 23** An example of template for a `ossExportExternal.phs` script.

---

```
«*Initialize a few date variables before...*»
«SQLOUTPUT»
  «INCLUDE
ossbo:ProcExport.phs#ExportTransBegin;_Reference="PMA«azRefTransac»";
_PostingDay=«_D»;_PostingMonth=«_M»;_PostingYear=«_Y»;
_ValueDay=«_D»;_ValueMonth=«_M»;_ValueYear=«_Y»;
_IdUser=«#SQLUSERID#»»
  «SQLOUTPUT»
  «INCLUDE
ossbo:ProcExport.phs#ExportPostingTag;_Reference="PMA«azRefTransac»";
_IdAccount=«idAccount»;_Credit=«mCred»;_Debit=«mDeb»;_Label=«azLabel»»
  «/SQLOUTPUT SELECT(*) FROM PAYMENT_DATA WHERE idTransac=«idTransac»»
  «INCLUDE ossbo:ProcExport.phs#ExportTransEnd»
«/SQLOUTPUT SELECT idTransac, azRefTransac, <etc.> FROM PAYMENT_DATA
<conditions>»

«IF #SQLSTATUS#==0 AND @toi(_SqlStatus)==0»
  «SYSTEM ECHOLOG EXPORT FAILED IN A !»
«ELSE»
  «SQL UPDATE PAYMENT_DATA SET idExport=«%d;_ExportId» WHERE
<conditions>»
«ENDIF»
```

---

At the end, the success of the export is tested ; if they were done successfully, it is possible to update the relevant table of the add-on with the identifier of the export, `_ExportId`.

### 2.10.9.1.3 `ossExportValidate.phs`

As a value of 1 is expected for a Shop export to be validated, this script simply consists in

```
RETURN 1
```

### 2.10.9.1.4 `ossExportCancel.phs`

This script, as launched when the export is canceled, may consist in using `_ExportId` to identify the export, and act on add-on A's related table as a consequence, for instance setting there its value back to 0 :

```
«SQL UPDATE PAYMENT_DATA SET idExport=0 WHERE idExport=«%d;_ExportId»»
«IF #SQLSTATUS#==0»
  «SYSTEM ECHOLOG EXPORT CANCELLATION FAILED IN A !»
  «RETURN 0»
«ELSE»
  «RETURN 1»
«ENDIF»
```

Whether this operation was successfully done or not, a value of 1 or 0 is returned to the Shop.

## 2.10.10 Other shop-related oss- scripts

The following table sums up those non-object-specific, mostly related to events such as login, logout, etc.

Script in add-on A	Function
<code>ossModuleRegister.phs</code>	Identifies the add-on (mandatory, 2.3.1.1)
<code>ossModuleInit.phs</code>	Registers the add-on's objects (2.10.2)
<code>ossModuleLogin.phs</code>	Will be launched each time a user logs in
<code>ossModuleLogout.phs</code>	Will be launched each time a user logs out
<code>ossModuleHeadline.phs</code>	Outputs an event control on user's desktop (2.8)
<code>ossModuleLoad.phs</code>	Is triggered when main page's menu is loaded
<code>ossModuleMessage.phs</code>	Can be used to display a message on main page (old interface, obsolete)
<code>ossModuleObjectType</code>	Returns the list of object types (obsolete)